國立臺灣大學電機資訊學院電機工程學系

博士論文

Department of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Doctoral Dissertation

# 基於疊層網路之資訊散播技術

# Overlay System Support for Information Dissemination

吳齊人

Chi-Jen Wu

指導教授：陳銘憲 博士、何建明 博士

Advisor: Ming-Syan Chen, Ph.D. and Jan-Ming Ho, Ph.D.

中華民國一百零一年七月
July, 2012

# 摘要

由於資通訊產業的發展和社群網路的普及應用，使用者生成內容(UGC)的概念已然成爲一種新的數位媒體內容生成的方式。而在各種資訊散播方式中，發布/訂閱模型(Publish/subscribe paradigm)是最適合做爲使用者生成內容的資訊散播方式。因此在這篇論文中，我們探討了以疊層網路爲基礎的資訊散播技術。我們對於三個在疊層網路上資訊散播的根本問題提出可行的解決方案。第一個問題是如何在網際網路中快速散播一個大檔案給多個訂閱者。針對此問題我們提出了Bee協定，在此協定中以縮小最大散播時間爲最大考量，企圖使每一個訂閱者都能在短時間內接收到檔案。Bee協定是一個分散式的疊層網路架構，每一個節點只須知悉其鄰居節點的資訊就可以互助合作傳送檔案。模擬實驗結果顯示Bee協定可以在異質性網路中達到良好的最大散播時間，並且在Flash crowd pattern下依然能保持高速散播能力。第二個問題爲在異質性網路中如何建構一棵疊層網路樹能最佳化整體系統的最大資訊散播時間。我們提出了一個新穎的LSBT模型來解決此最佳化問題。在LSBT模型中，對每一個上傳連線(Uplink)定義了一個基本傳送速率$r$，所有的節點只能用$r$的整數倍來散播檔案。基於LSBT模型，我們提出了一個O($nlog^2n$)的演算法來選擇最佳的傳送速率$r$並建構出一棵最佳的LSBT。模擬實驗結果顯示LSBT在最大資訊散播時間及時間複雜度上都遠勝目前最好的演算法。第三個問題如何建構一個高延展性及高效率的行動現狀資訊散播網路架構。行動裝置的普及讓社交網路應用服務已成爲目前最重要和熱門的網路應用，也帶來了創新的網路營運模式。而社交網路服務中最爲基礎的技術就是行動現狀資訊服務(Mobile Presence Service)，它必須讓使用者能及時的搜尋其朋友目前的現狀並通知朋友本身的現狀。因此我們針對需要面臨大規模使用者的社交網路服務設計了一個俱延展性及高效率的伺服器網路架構，稱爲PresenceCloud。PresenceCloud俱備了高效率和高延展的搜尋演算法，其減少了搜尋現狀和通知朋友所需要的訊息量，並使現狀訊息傳遞所需的時間能保持在使用者可以接受的延遲。我們分析PresenceCloud並建構了模擬器來衡量其效能。實驗結果顯示PresenceCloud可以達成相當的搜索的滿意度和搜查效能。

本論文主要探索了三個以疊層網路爲基礎的資訊散播根本問題，然而隨著使用者生成內容發展和應用的多樣化，資訊散播技術在網際網路的應用將會越來越重要，而各式各樣資訊散播的需求也會隨之而生。

# ABSTRACT

SINCE the rapid advances of ICT and the increasing popularity of social networking applications, User Generated Content (UGC) has become a new media content production circles (Web 2.0). The publish and subscribe paradigm is most suitable for the design principle of UGC applications. Thus we are interested in the issues of developing overlay architectures and techniques to support dissemination of information in publish and subscribe paradigms. Specifically, we focus on understanding and investigating the following fundamental questions of information dissemination on overlay systems **Q1)** *How to rapidly disseminate a large-sized file to many subscribers in the Internet when the publisher releases the file*? **Q2)** *What is the relation between a single overlay tree with a fixed uplink rate and the broadcast operation itself, and how to construct a single overlay tree that minimize the maximum completion time in heterogeneous networks*? **Q3)** *Does there exist any other messages dissemination service that significantly outperforms those based distributed hash tables (DHT) both on scalability and efficiency for mutable and dynamic information, such as mobile presence messages?*

To address the **Q1)**, we present the Bee protocol, which is a best-effort peer-to-peer file delivery protocol aiming at minimizing the maximum dissemination time for all peers to obtain the complete file. Bee is a decentralized protocol that organizes peers into a randomized mesh-based overlay and each peer only works with local knowledge. We devise in Bee protocol a slowest peer first strategy to boost the speed of dissemination, and a topology adaptation algorithm that adapts the number of connections based on upload bandwidth capacity of a peer. Moreover, Bee is designed to support network heterogeneity and deal with the flash crowd arrival pattern without sacrificing the dissemination speed.

We then explore a novel model, named LockStep Broadcast Tree (LSBT), of big data broadcasting in heterogeneous networks for **Q2)**. The main idea in our LSBT model is to define a basic unit of upload bandwidth, $r$, such that each node uses only integer multiples of $r$ in broadcasting. We show that the optimal uplink rate $r^*$ of LSBT is either $c/2$ or $c/3$ in in homogeneous network environments. Then we present an $O(n \log^2 n)$ algorithm to select the optimal uplink rate $r^*$ and to construct an optimal LSBT for heterogeneous environments.

Finally, we examine the intrinsic scalability problem of server-to-server architectures for mobile presence services for **Q3)**. To address the problem, we propose a scalable and efficient server architecture, called PresenceCloud, which enables mobile presence services to support large-scale social network applications. PresenceCloud organizes presence servers into a quorum-based server-to-server architecture for efficient presence searching. It also leverages a directed search algorithm and a one-hop caching strategy to achieve small constant search latency. We analyze the performance of PresenceCloud in terms of the search cost and search satisfaction level.

In summary, we make offers to study the three fundamental questions of overlay systems we mentioned for designing better information dissemination systems in publish and subscribe paradigms. However, the truth is that there is *no one fit all solution* for information dissemination services, future information dissemination services of publish and subscribe paradigms shall call for a generic, resilient, efficient and reliable platform for the demand of **everyone** in the Internet.

# ACKNOWLEDGEMENTS

*Thanks for Dr. Jan-Ming Ho and Prof. Ming-Syan Chen taking me under their wing at a time. Since I have profited immensely from their wisdom and high standards.*

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES



# CHAPTER 1
# Introduction

"*Content is King*."

–William Henry Bill Gates III - 1996

USER Generated Content (UGC) has arisen in content publishing and has become a new media content production circles (Web 2.0) in the Internet, with the rapid advances of ICT and the increasing popularity of social networking applications and mobile devices. UGC is a successful principle and tremendously popular, it is used for a wide range of applications available in modern Internet. The Internet technologies, including wired and wireless, have made UGC information sharing a commonplace. The UGC applications include digital video, blogging, podcasting, social networking, mobile phone photography, and the novel UGC applications are created on the way. Moreover, human social incentives are the most common form of implicit incentives. These incentives make a user to maintain his/her activities as a vivacious member of the social community, such as Google+, Facebook or Twitter. Many startup companies in the Internet industry, such as Instagram[1], have foreseen the increasing demand of UGC and have made profits from the concept of UGC. Meanwhile, another point worth noting is Machine Generated Content (MGC), thus the information is generated or collected by machines. With the rapid development of cloud computing and the increasing ubiquity of Internet of Things (IoT), this new trend might bring us whole new applications in the next generation Internet. As a result, today's Internet is now replete with exabytes of UGC and we can envision that more and more information will be produced from UGC and MGC in the future Internet.

The trend toward fast increased demands of UGC applications, content distribution networking has recently become an important research topic and is receiving a significant amount of interest from both academic and industrial environments. However, most

[1] Instagram is a free photo sharing application launched in October 2010 that allows users to take a photo and share it on one of social networking services. Facebook acquired Instagram in April 2012.

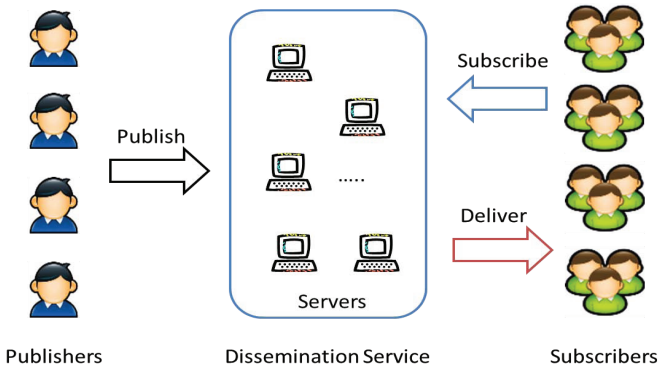



Figure 1.1: An illustration of high level view of a publish and subscribe system

content distribution systems in human sociality are based on the publish and subscribe paradigm, for example, news papers or magazines. In the publish and subscribe paradigm, subscribers send requests to publishers. Then publishers reply to subscribers' requests. Most importantly, the publish and subscribe paradigm is most suitable for the design principle of UGC applications. It enables everyone can publish his/her content, and also let each one can subscribe the interested and needed information. Thus, in the cyberspace, the main specificity of the publish and subscribe paradigm inheres in the way the information is disseminated from the source (publisher) to the destinations (subscribers). Figure 1.1 illustrates those concepts of a publish and subscribe system in the cyberspace. A publish and subscribe system consists of three major components [1]: *i*) *publishers* are the producers of information that are published to the dissemination service. *ii*) *subscribers* interest in the information and they register their interest information by the dissemination service. *iii*) *dissemination service* is the main dedicated infrastructure that becomes the interface between publishers and subscribers. Specifically, it achieves the dissemination of information by routing information messages published from publishers towards the subscribers interested. In the next part of the chapter, we briefly describe the dissemination service of the publish and subscribe paradigm and its evolution.

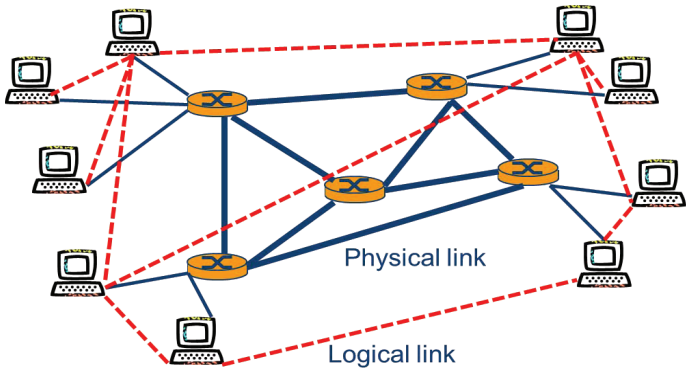



Figure 1.2: An illustration of an overlay system

## 1.1 Dissemination in Publish and Subscribe Paradigm

In this section, we briefly describe the evolution of the dissemination service, and the various system architectures around which a disseminate service can be built.

**Centralized C/S architecture** is the simplest one of dissemination services in the publish and subscribe paradigm. However, for scalability and system churn issues, it is not suitable for large scale systems and may only be viable in small networks or in systems with a very limited number of user.

**Multicast based infrastructures** Native IP multicast is an efficient and scalable solution for dissemination of information. A multicast based protocol has the advantage of providing efficient dissemination of information in terms of latency and throughput. However, a number of considerations, including scalability, reliability and business policy of ISPs have limited the widespread success and availability of IP multicast across the Internet infrastructure. For that reason, the implementation of a large scale publish and subscribe system using existing multicast is strongly limited by the rather limited commercial deployment of multicast infrastructures. Since the native IP multicast solution has not generally received widespread deployment, a number of efforts focus on building information dissemination service by the concept of using overlay systems.

**Overlay system** With the rapid advancement of ICT technology, much more aggregate information, network bandwidth and computing resources are available from clients than from a limited number of centralized servers. Thus the concept of overlay system has become a main trend in the design of Internet protocols. An overlay system is a layer

of virtual network topology on top of the Internet, as shown in Figure 1.2, which directly connections to clients. Overlay systems provide us with the following advantages to better utilize the increasingly growing information and resources on the Internet. It allows both system developers and users to design and implement their own communication protocols on top of the Internet. The end nodes in overlay systems are highly connected to each other due to flexible routing of today's Internet. Besides, overlay systems are highly decentralized. Thus, highly flexible is the most attractive feature of overlay systems.

The main research focus of this dissertation is to develop and to evaluate overlay system techniques to support information dissemination services in publish and subscribe paradigms.

## 1.2 Challenges and Issues

Designing information dissemination services is a complex task. In this dissertation, we aim at addressing these challenges that are common to the design of overlay system for information dissemination services in the context of building efficient, scalable publish and subscribe applications. The specific research challenges and issues are described as follows:

1) **Scalability** is the ability of a information dissemination system to handle growing amount of users in a capable manner or its ability to be enlarged to accommodate the system growth. From the view of publishers, the information dissemination system must scale not only for the number of subscribers, but for the number of server nodes. Since a popular information dissemination system with tens of thousands of subscribers are common, the overlay system must scale to support such large sizes, and the overhead associated must be reasonable even at large scales.

2) **Efficiency** The overlay system constructed must be efficient from both the service provider and the application users perspectives. For a lager-size file dissemination, high bandwidth and rapid transmission are simultaneously required. However, given that applications are interactive, such as social networking communication, a communication delay of a few hundreds of milliseconds can be tolerated.

3) **Churn** can be measured in various dimensions, such as system and information.

In the view of a system, it refers to participants (eg., users) turnover in information dissemination systems or server nodes entering and departing the system at arbitrary points in time. This challenge is to maintain system performance among the dynamic participants and it is crucial for the success of large scale information dissemination. On the other hand, information churn can refer to information that is more mutable and dynamic, such as presence information. For this case, the challenge is to delivery the message to the subscribers as soon as possible.

4) **Data size** does matter to information dissemination systems, it also dominates the design of overlay systems. There are a number of trends associated with Big Data and cloud computing that are clearly beginning to emerge. In domains as diverse as telecommunications, financial services, retail and web applications, large data volumes have been a normal part of daily operations for at least a decade. Thus, big data is a challenge for information dissemination systems in an enterprise (in a data center). In addition, how to disseminate a small-size message to a large-scale subscribers is also an non-trivial problem.

5) **Heterogeneity** always makes problems more difficult to solve. In the Internet, node has heterogeneity characteristic, including computing power, uplink capacity, store capacity, etc. Therefore, it is essential to augment an overlay system that distributes the communication load among nodes in the system according to their resource availabilities. Ideally the load assigned to each node should be proportional to its capabilities. How to prevent nodes become bottlenecks and exploit heterogeneity to improve the efficiency of the system is a great challenge in overlay system design.

## 1.3 Overview of the Dissertation

### 1.3.1 Rapid Data Dissemination

Currently, a number of efforts focus on building P2P content delivery architectures [2–6] to address the data dissemination problem. BitTorrent system [2] is one of the pioneers of the file dissemination systems, and has become a prominent Internet application, both in terms of user popularity and traffic volumes [7]. However, BitTorrent is designed to

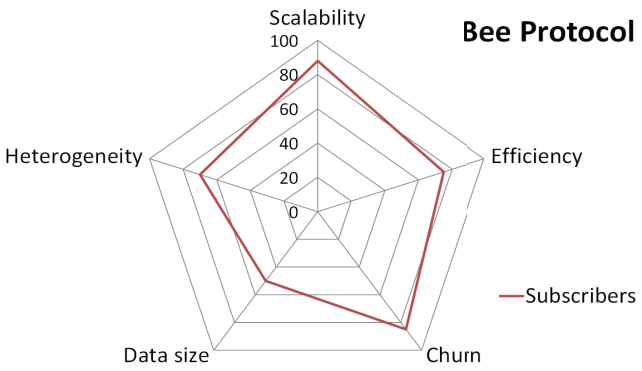



Figure 1.3: A performance illustration of Bee Protocol

minimize the dissemination time of each peer egoistically.

Thus, in CHAPTER 2 of this dissertation, we focus on understanding and investigating the following question: *How to rapidly disseminate a large-sized file to many subscribers in the Internet when the publisher releases the file*? Specifically, we proposed a decentralized data dissemination protocol to minimize the data dissemination time. The experimental results confirm the analysis in showing that the new design can significantly reduce the overall download time and its performance should be scalable to a large number of users. Bee is more suitable for a global commerce company to distribute content (software) to end-customers. Our Bee protocol should be robust to user churn and be adaptive heterogeneous networks, it is also simple to be implemented and deployed. In Figure 1.3, we show our design of Bee Protocol can achieve system performances in terms of the five issues we mentioned.

### 1.3.2 Efficient Big Data Broadcasting

Big-Data Computing, a new critical challenge has sparked the major research issues that are reshaping ICT industry and scientific computing in the past few years [8]. Currently, both ICT industry engineers and scientific researchers may have to deal with petabytes of data sets in the cloud computing paradigm [9]. Thus the demand for building a distributed service stack to efficiently distribute, manage and to process massive data sets has risen drastically. There are many of significant issues for developing data-parallelism applica-

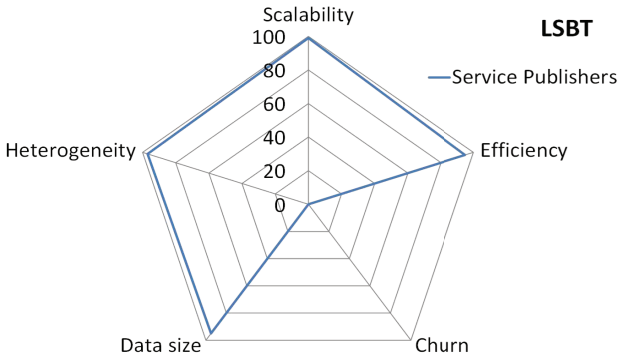



Figure 1.4: A performance illustration of LSBT algorithm

tions, such as, the effective strategy for data decomposition, load balancing on processing nodes, and data transmissions between a large set of nodes [10]. In particular, for big-data computing, data transmission operations may become one of most significant issues to reduce the job completion times.

Thus, we focus on investigating the following questions: *What is the relation between a single overlay tree with a fixed uplink rate and the broadcast operation itself, and how to construct a single overlay tree that minimize the maximum completion time in heterogeneous networks*? Specifically, we propose the LockStep Broadcast Tree (LSBT), which models the original big-data broadcasting problem to support heterogenous networks by constructing an efficient pipeline broadcasting tree. We also design a novel polynomial-time algorithm to select the optimal uplink rate of $r^*$ for building an optimal LSBT. Specifically, given a data delivery deadline, LSBT may be possible to determine that can the network system meet the deadline or what is the possible deadline for the delivery job. This advantage can maximize the performance of collaborative applications in datacenter networks. In Figure 1.4, we show our design of LSBT algorithm can achieve system performances in terms of the five issues we mentioned.

### 1.3.3 Mobile Presence Dissemination

A mobile presence service is an essential component of social network services in cloud computing environments. The key function of a mobile presence service is to maintain

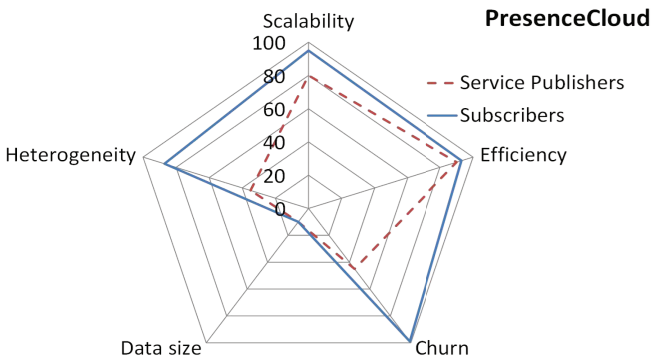



Figure 1.5: A performance illustration of PresenceCloud

an up-to-date list of presence information of all mobile users. The presence information includes details about a mobile user's location, availability, activity, device capability, and preferences. The service must also bind the user's ID to his/her current presence information, as well as retrieve and subscribe to changes in the presence information of the user's friends. Currently, more than five billion people use social network services on the Internet [11]. Given the growth of social network applications and mobile network capacity, it is expected that the number of mobile presence service users will increase substantially in the near future.

Hence, we focus on studying the following questions: *Does there exist any other messages dissemination service that significantly outperforms those based distributed hash tables (DHT) both on scalability and efficiency for mutable and dynamic information, such as mobile presence messages?* Specifically, we propose the design of PresenceCloud, a scalable server-to-server architecture that can be used as a building block for mobile presence services. The rationale behind the design of PresenceCloud is to distribute the information of millions of users among thousands of presence servers on the Internet. We analyze the performance complexity of PresenceCloud and two other architectures, a Mesh-based scheme and a Distributed Hash Table (DHT)-based scheme. Through simulations, we also compare the performance of the three approaches in terms of the number of messages generated and the search satisfaction which we use to denote the search response time and the buddy notification time. The results demonstrate that PresenceCloud achieves major performance gains in terms of reducing the number of

messages without sacrificing search satisfaction. Finally, Figure 1.5 shows our design of PresenceCloud can achieve system performances in terms of the five issues we mentioned.

## 1.4 Organization of the Dissertation

The rest of this dissertation is organized as a series of chapters. In each of these chapters, background, and system models are given before the core technical content is described. The specific contributions are given in the introduction part of each chapter, whereas the related work in the literature is reported at the end of each chapter. Concretely, this thesis is composed of the following chapters.

In CHAPTER 2, we are interested in the data dissemination problem in which the maximum dissemination time of all the peers instead of the dissemination time of individual peers is minimized. We present a decentralized protocol, called Bee, to address the data dissemination problem.

In CHAPTER 3, we discuss the Big data broadcasting problem both in practical and theoretical ways. We consider a heterogeneous network system with nodes, *i.e.*, nodes may have different uploading capacities. The Big data broadcasting problem is about how to let these nodes efficiently obtain the entire bulk data cooperatively, and how to minimize the total transmission time in each node.

As for CHAPTER 4, we explore the relationship between distributed presence servers and server network topologies on the Internet, and propose an efficient and scalable server-to-server overlay architecture, called PresenceCloud, to improve the efficiency of mobile presence services for large-scale social network services. Finally, we conclude this dissertation with a summary of main research results in CHAPTER 5.

# CHAPTER 2
# Rapid Data Dissemination: A Peer-to-Peer Approach

"*Simple is beautiful.*"
–Unknown

HOW to rapidly and efficiently distribute a large file across the Internet has become an interesting problem in the peer-to-peer (P2P) research community and some real applications, such as updating the software patches of Massively Multiplayer Online Games (MMOG) and operation systems in a flash crowd arrival pattern. Suppose that a large-sized critical data is initially held at a single server and we have to disseminate it to other $N$ peers rapidly, how can we minimize the time it takes for all peers to have the complete data? This problem is very practical for the time-critical applications in the Internet. Let us consider the scenario that an enterprise of online game, e.g. Blizzard Entertainment needs to upgrade the patches of end users. The dissemination time of all users will become the important key metrics for all users and the business company.

Native IP multicast is an efficient and scalable solution for the data dissemination problem [12]. However, a number of considerations, including scale, reliability and business policy of ISPs have limited the widespread success and availability of IP multicast across the Internet infrastructure. Since the native IP multicast solution has not generally received widespread deployment, a number of efforts focus on building P2P content delivery architectures [2–6] to address the data dissemination problem. The main idea is that a source can transmit data to a set of peers by TCP connections, and these peers across the Internet act as intermediate routers to deliver data over a predefined network, such as a mesh or multiple trees at the application level.

Currently, a number of efforts focus on building P2P content delivery systems [2–

4,6] to address the data dissemination problem. BitTorrent [2] is one of the pioneers of the file dissemination systems, and has become a prominent Internet application, both in terms of user popularity and traffic volumes [7]. BitTorrent is designed to reduce the load from congested servers and improve the download time of software. However, in such emergency conditions, such as dissemination of the critical patch updates for security or deployment of scientific data between institutions, e.g. CERN, all participants have to obtain data fast. For this reason, all participants should be cooperative for the fastest dissemination, the fairness mechanism, such as tit-for-tat scheme, are often unnecessary. On the other hand, other recent studies (including Slurpie [3], Bullet [4], SplitStream [5] and Crew [6]) have proposed to construct and maintain an overlay network of multiple trees or a random mesh to deliver data from a single server. In these P2P architectures, data are usually divided into $M$ parts of equal size, each being called a chunk or block. A peer may download any one of these blocks either from the server or from a peer who has downloaded that chunk. Besides, many researchers, including [13–15], have studied the performance of these P2P architectures, and shown that the P2P architecture is both efficient and scalable, even it lacks a centralized coordination and scheduling mechanism. In the *best-effort* service concept, a peer allocates bandwidth for all its neighbors and attempts to serve all of them without idle time. Basically, Bee is similar to the BitTorrent system, but with a different purpose.

In this chapter, we are interested in the **time-critical** data dissemination problem in which the maximum dissemination time of all the peers instead of dissemination time of individual peers is minimized. We begin by giving a formal definition of the data dissemination problem, and introduce the lower bound of the maximum dissemination time. We then present a decentralized protocol, called Bee, to address the time-critical data dissemination problem. Bee is a *best-effort* and and cooperative designed protocol to increase throughput of the system and individual peer. In the Bee protocol, peers are self-organizing into a random mesh and download blocks from neighboring peers. Moreover, Bee peer leverages a slowest peer first strategy and a topology adaptation algorithm to maximize the speed of dissemination. Under the slowest peer first strategy, a peer always transmits blocks to the neighbors that have the fewest number of downloaded blocks, i.e., the slowest neighbors. To decide the number of connections of a peer, Bee

protocol embeds a topology adaptation algorithm based on peer's upload bandwidth. Our experimental results show that our Bee protocol can approach the lower bound of the data dissemination problem for both homogenous and heterogeneous network environments. In addition, we also show that the Bee system does not sacrifice too much fairness while pursuing the dissemination speed. For example, those peers with higher upload bandwidth only have a small amount of overhead in the dissemination time as compared with BitTorrent. To the best of our knowledge, this work is among the first one to systematically study the effects of distributed P2P architecture with respect to minimizing data dissemination time in data networks. However, we also use fairness in terms of the amount of upload data and dissemination time of the high-bandwidth peers as performance indices. Specifically, we make the following contributions in this chapter:

1) we derive the lower bound of the dissemination time for the data dissemination problem. Although many works [16–18] also studied this problem, most of them focus on the theory deduction, we differ from them on analyzing how to design a protocol to approach the lower bound.

2) we proposed a decentralized data dissemination protocol to minimize the data dissemination time. The experimental results confirm the analysis in showing that the new design can significantly reduce the overall download time and its performance should be scalable to a large number of users. Bee is more suitable for a global commerce company to distribute content (software) to end-customers.

3) our Bee protocol should be robust to user churn and be adaptive heterogeneous networks, it is also simple to be implemented and deployed.

The rest of the chapter is organized as follows. In next section, we discuss related work. In Section 2.1, we first define the data dissemination problem. Section 2.2 describes an overview and design details of the Bee protocol. Section 2.3 explains our simulation methodology and presents the performance results of the simulation study. We conclude this chapter with a summary of main research results in Section 2.5.

## 2.1 Data Dissemination Problem

In this section, we formally define the data dissemination problem, and provide the lower bound of the problem. Let us consider the problem of disseminating a file $F$ to a set of $n$ peers, $\mathcal{N} = \{1, 2, \ldots, n\}$. We also assume that a peer leaves the system once completely receiving the file. Let $S$ be the server (which is called a "seed" in the rest of this chapter) that has the file $F$ in the beginning, and let $Size(F)$ denote the size of file $F$ in bytes. Each peer $i \in \mathcal{N}$ in this system has its upload capacity $U_i$ and download capacity $D_i$. And $U_s$ represents the upper bound of the upload bandwidth utilization of seed $S$. We also assume that $U_i \leq D_i$, to model the state-of-the-art Internet technologies, such as ADSL or Cable modems. Due to the asymmetric nature of these network technologies, $D_i$ is usually 3 to 5 times higher than $U_i$ in practice. Let $t_i(F)$ denote the time it takes for peer $i$ to receive the complete file $F$. Note that $t_i(F)$ denotes the time interval starting at the time peer $i$ sends its request to the server and ending at the time it receives the entire file $F$. Before formally defining the problem, we define the following two performance metrics first.

**DEFINITION 2.1.1 (Average Dissemination Time)**

$$ADT(F) = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} t_i(F).$$

**DEFINITION *2.1.2 (Maximum Dissemination Time)***

$$MDT(F) = \max\{t_i(F)\}, i \in \mathcal{N}.$$

Assume that the server and all $n$ peers exist in the system from time $t = 0$. Then $MDT(F)$ is the time it takes for all peers to finish receiving the complete file $F$. Now we define the data dissemination problem as follows.

**DEFINITION 2.1.3 (Data Dissemination Problem)** *Given a server and $n$ peers in a system, and each peer $i$ has the upload capacity $U_i$ and download capacity $D_i$, where*

$i = \{1, 2, \ldots, n\}$, *the problem is to find a transmission scheme* $\mathcal{M}$ *to minimize the* $MDT(F)$. *According the definition 2, the problem can be formulated as a min-max problem as follows.*

$$\min\{MDT(F)\}. \tag{2.1}$$

### 2.1.1 Ideal Dissemination Time (Lower Bound)

In this section, we focus on studying the lower bound of the dissemination time, which is also referred to as *ideal dissemination time* in this chapter. Let us denote the actual amount of data uploaded by peer $i$ as $f_i$, where $f_i \leq U_i \times t_i(F)$ and those peer $i$ receives in return as $r_i$, where $r_i \leq D_i \times t_i(F)$. Without loss of generality, we assume that, the total amount of download data is equal to the total amount of upload data for the seed and all peers. Hence, we have the following equation.

$$f_s + \sum_{i=1}^{n} f_i = \sum_{i=1}^{n} r_i. \tag{2.2}$$

Since we are interested in estimating the lower bound of the dissemination time, we assume that upload capacity of each peer $i$ is assumed to be fully utilized, i.e., we have $f_i = U_i \times t_i(F)$. Besides, the total amount of download bandwidth must be equal to $n \times Size(F)$, because all peers have the entire file $F$ at the end. Then, we can extend the Eq. 2.2 as follows to deal with the ideal dissemination time for the general case.

$$U_s \times t_s(F) + \sum_{i=1}^{n} U_i \times t_i(F) = n \times Size(F). \tag{2.3}$$

Here, we have a min-max problem with its objective function in Eq. 2.1 subject to the constraints given by Eq. 2.3. Since the constraint is a linear equation, or more specifically, a hyperplane in $(n+1)$-dimension, and the optimal solution for the constrained optimization problem can be obtained if and only if when all $t_i$ are the same, i.e.,

$$t_s(F) = t_1(F) = t_2(F) = \cdots = t_n(F). \tag{2.4}$$

Applying this results to Eq. 2.3, we then have a lower bound of $MDT(F)$, denoted by $\overline{T}(F)$, for file $F$, as follows.

$$\overline{T}(F) = \frac{n \times Size(F)}{U_s + \sum_{i=1}^{n} U_i}.$$

Thus, we have the **LEMMA** 2.1.1.

**LEMMA 2.1.1** *Let $T^*(F)$ denote the $MDT(F)$, of a feasible schedule of the data dissemination problem for a given file $F$. Then we have:*

$$T^*(F) \geq \max\left\{\overline{T}(F), \frac{size(F)}{U_s}, \frac{size(F)}{\min\{D_i\}}\right\}, \tag{2.5}$$

*where the right-hand right of Eq. 2.5 is the lower bound of the dissemination time in any algorithm for the data dissemination problem.*

## 2.2 Bee Design

In this section, we present the Bee protocol to approach the ideal dissemination time. In a Bee system, the content (file) is divided into many fix-sized blocks $B_i, i = \{1, 2, \ldots, m\}$, which is the smallest transfer unit in the system. We chose 256KB for the block size, which is the same as that used in most other P2P protocols. In the following, we start with an overview of the Bee protocol, followed by the detailed descriptions of its various components.

### 2.2.1 An Overview of Bee

At a concept of overview, Bee constructs a random mesh overlay among a set of peers. Fig. 2.1 illustrates the scheme of a Bee system. Suppose that a large content is announced from a single server, and particularly we assume that $U_s > U_i$ in the system, and a lot of peers want to download the content at the same time. Note that it is reasonable, for example, an enterprise of online game, ex Blizzard needs to distribute the critical patch to

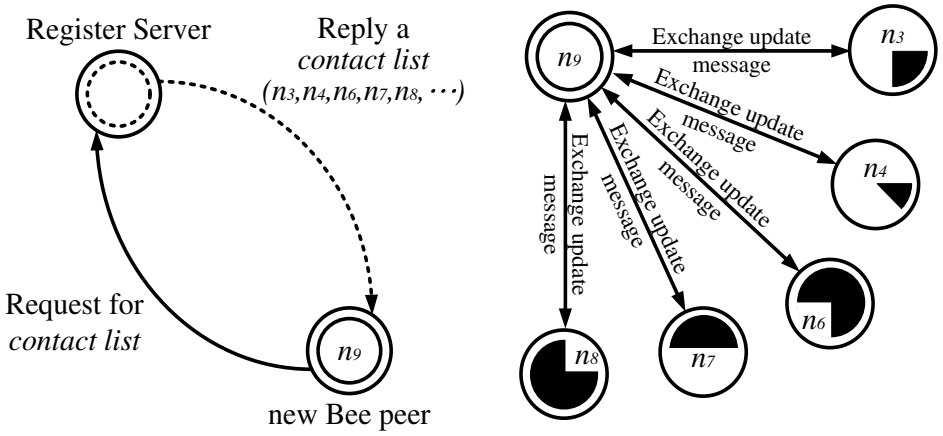



Figure 2.1: A scheme of a Bee system

end users. The download bandwidth of Blizzard's servers should be large than end-users. Each peer gets into contact with a centralized well-known register server and retrieves a *contact list* of an uniform random subsets of all peers in the system. The size of contact list is a small constant, say 160.

Based on the contact list, the peer discovers other peers, and exchanges update messages with them. The update message contains a *bitstring* about which blocks are available, and the bitstring can be used to coordinate the block requesting decisions without global information. After exchanging update messages, a peer could send requesting block messages to all peers in the contact list, and download blocks while uploading blocks it owns to other peers simultaneously. In a peer to peer network, peers may leave the system dynamically. In order to maintain connectivity of the overlay network, peers in Bee system will periodically, say 30 seconds, ask the register server to obtain a new contact list of replacement peers.

The key components in the Bee system include (1) a **slowest peer first strategy** for selecting peers to upload blocks, (2) a **chunk selection strategy** for requesting blocks, and (3) a **topology adaptation algorithm** for adapting the number of connections according to a peer's network capacity. The detailed descriptions of these components are presented as follows.

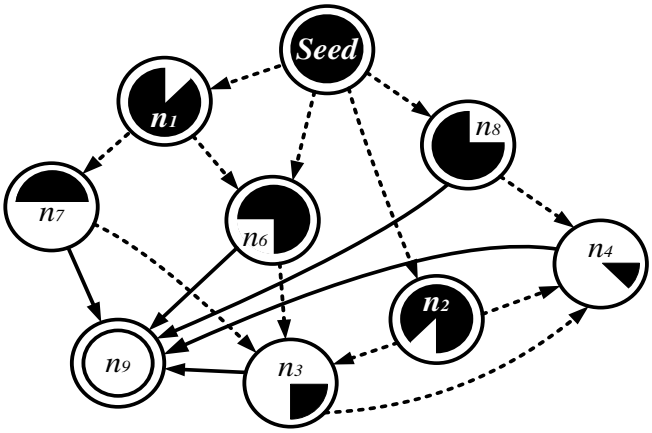



Figure 2.2: An illustration of the slowest peer first strategy: The shaded content within a peer represents the percentage of the file that a peer has downloaded. Using the slowest peer first strategy, a peer always forwards blocks to the slowest peer among its contact list.

### 2.2.2 Slowest Peer First Strategy

In this section, we describe the slowest peer first strategy and the procedure that each peer performs. Overall, a good peer selection approach would be one which neither requests peers to maintain global knowledge, nor communicate with a large number of peers, but the one which is able to find peers having blocks the peer needs. Hence, our focus is the development of a decentralized mechanism, within which each peer learns its nearby peers' statuses (with local knowledge) and selects a suitable peer to upload blocks in order to minimize the maximum dissemination time of the system.

The design principle of the slowest peer first strategy is to keep all the upload capacity of peers full of data. This means that a peer always can find some peers to upload blocks to exhaust its upload capacity. Based on the slowest peer first strategy, a peer $i$ always picks a slowest downloading peer among the contact list of the peer $i$, where the slowest downloading peer is the peer that has the least number of blocks. Consequently, the peer $i$ always can upload blocks to the picked peer, because there is a high probability that the peer $i$ has some blocks that the picked peer does not have. In this point of view, this peer selection strategy makes our Bee protocol to be able to utilize the upload bandwidth of each peer as much as possible.

The operation of the slowest peer first strategy is very simple but efficient. We use

an illustration of the slowest peer first strategy in Fig. 2.2 to present this simple idea. After a peer joins into a Bee system, it periodically sends the requesting block messages to the peers in the contact list for downloading the blocks it lacks. When a peer starts to upload blocks to other peers, it maintains a *working set*, and the peer tries to use its network capacity as much as possible to upload blocks to the peers in its working set. The working set is a set of peers selecting from the contact list over a period of time. The size of working set is adapted by a topology adaptation algorithm that we will discuss later. We show the pseudo code of the slowest peer first algorithm as follows.

**Algorithm 2.1** Slowest Peer First Strategy

```
1: BEGIN
2: WorkingSet[] ← empty
3: for j ← 0 to Sizeof(WorkingSet[]) do
4:    Pick the slowest peer p ∈ Contact List
5:    WorkingSet[j] ← peer p
6:    j ← j + 1
7: end for
8: return WorkingSet[]
9: END
```

After a peer joins into a Bee system, it periodically sends the requesting block messages to the peers in the *contact list* for downloading the blocks it lacks. When a peer starts to upload blocks to other peers, it maintains a *working set*, and tries to exhaust its network capacity as much as possible to upload blocks to the peers in its working set. The working set is a set of peers selecting from the contact list over a period of time. The size of working set is adapted by a topology adaptation algorithm that we will discuss in the Section 2.2.4.

The advantage of the slowest peer first strategy is to enable peers to download blocks at the same progress approximately, and this strategy can significantly diminish the $MDT$ that we defined in Section 2.1. The disadvantage of the strategy is that the faster downloading peers may be delayed for the slower downloading peers. However, our goal is to minimize the maximum dissemination time of the system. Let us think if some high capacity peers finish downloading and leave the system early, then the maximum dissemination time of the system should be prolonged undoubtedly. Hence, this problem can be addressed by the proposed slowest peer first strategy fundamentally. Note that, our

slowest peer first strategy only makes the peers download the file at the same progress approximately, it dose **not** require all peers leaving at the same time, every peer can immediately depart the Bee system at an arbitrary time.

### 2.2.3 Chunk Selection Strategy

Once a peer establishes connections with its neighboring peers, it needs to determine which blocks to request from which peers, based on the local knowledge (the available blocks in all peers among the contact list). The Bee protocol employs the local rarest first strategy for choosing new blocks to download from neighboring peers. The local rarest first strategy is proposed in BitTorrent protocol, and it can prevent the last block problem and increase the file availability in a BitTorrent system. The main advantage of the local rarest first strategy is to overcome the last block problem by favoring rare blocks. This strategy equalizes the file block distribution to minimize the risk that some rare blocks are lost when peers owning them fail or depart the system. Bharambe *et al.* [19] study the local rarest first strategy by simulations and show that this strategy can address the last block problem efficiently. Another advantage of the local rarest first strategy is to increase the probability that a peer is useful to its neighboring peers because it owns the blocks that others do not have, and thus it helps diversify the range of blocks in the system. For above reasons, the Bee protocol also uses the local rarest first strategy to select the requesting blocks. Moreover, applying this strategy, Bee can reduce the complexity of protocol communication, because each peer only needs to maintain local information of each block.

Next, we represent the computing model of the local rarest block strategy. Suppose that $B$ denotes the set of overall blocks in the file being distributed, and $G_i$ and $M_i$ are the set of blocks that peer $i$ has already received and is still missing, respectively (where $B = M_i \cup G_i$ and $M_i \cap G_i = null$). Similarly, $M_i \cap M_j \neq null$ means that the peers $i$ and $j$ both are missing at least a block $m$, and they try to find and download the block $m$. The requesting peer $r$ selects the rarest block $\hat{m} \in (G_j \cap M_r)$ among those that it misses and one of its neighbors, say peer $j$, held. The rarest block is computed from the number of each block that held by the neighbors of requesting peer $r$. More precisely, we use the

following computing function for the local rarest block strategy:

$\forall\, m \in (G_j \cap M_r)$, where $j$ belongs to *Contact List*

$$\mathcal{LRF}(\hat{m}) = \max \left\{ \sum_{i \in \textit{Contact List}} |m \cap M_i| \right\}.$$

The above function $\mathcal{LRF}(\hat{m})$ represents a computing model of the local rarest block strategy for a peer $r$. For each block $m \in (G_j \cap M_r)$, a peer $r$ calculates the number of neighboring peers missing the block, and it chooses the block with the maximum value for requesting download.

### 2.2.4 Topology Adaptation Algorithm

The design of Bee explicitly takes into account the capacity heterogeneity associated with each peer in the P2P network. Bee is designed to adapt to different network environments by a topology adaptation algorithm. In general, the available bandwidth estimation is a non-trivial problem in practical network applications, so it is hard to decide how many upload connections a peer should have in the Bee protocol. However, using a fixed number of upload connections will not perform well under a wide variety of peers' network capacities. Hence, the Bee protocol employs a simple flow control algorithm that attempts to dynamically change the maximum number of upload connections (indicated the size of the *working set* in Bee) according to the upload capacity of each peer.

For the sake of simplicity, we do not use the network bandwidth estimation techniques [20] to determine the precise upload capacity of each peer. Instead, we assume that a user can input a coarse-grained bandwidth estimate, such as the form ADSL, Cable, T3, etc, that provides an initial maximum upload capacity estimate, $U$. In addition, we assume that peers (including the seed) have limited upload/download bandwidth but the Internet backbone is assumed to have infinite bandwidth, as illustrated in Fig. 2.3. This assumption is reasonable, because the previous study [21] showed that the Internet backbone indeed has low utilization and the bottleneck almost happens at the parts near the end hosts. Based on the two assumptions, we can develop a simple topology adaptation

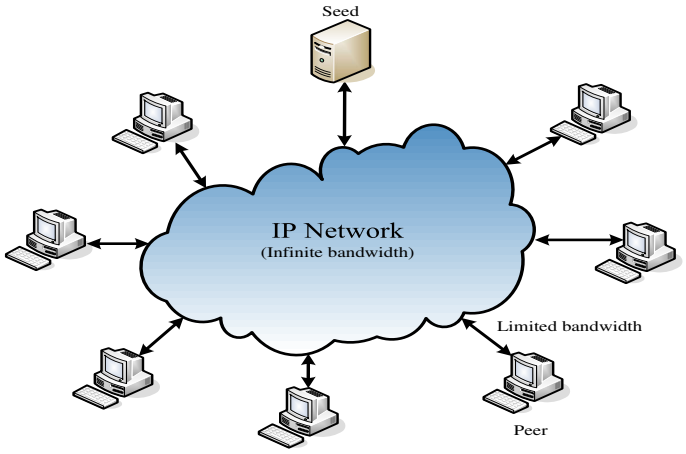



Figure 2.3: The simplified network-capacity framework

algorithm for the Bee protocol.

A simple adaptation approach is to set the upload rate for each upload connection to the same value, say rate $r$, for all peers. Hence, if a peer $i$ has maximum upload capacity of $U_i$, it establishes $k = \lceil \frac{U_i}{r} \rceil$ connections, where $r \leq U_i$, $\forall\ i \in \mathcal{N}$. So in the Bee protocol, each peer establishes $k$ concurrent upload connections among its working set, and intuitively a peer can quickly upload the blocks it holds to other peers. Note that the $k$ should be bounded by the size of the contact list.

The basic idea behind this approach is that by serving $k$ different peers simultaneously. Each peer can fully utilize its upload capacity and maximize its contribution to the system throughput. Our idea also fits the measurement results [22] show that high-bandwidth peers can contribute bandwidth to upload data to low-bandwidth peers and help reduce the maximum dissemination time of the system. For example, a peer with higher capacity might establish ten or more connections than a peer with lower capacity, it should contribute more upload bandwidth than a low-bandwidth peer do. However, this intuition is not always correct. When a smaller upload rate $r$ is chosen, the lower bandwidth dedicates to each connection, it might slow down the distribution rate for blocks. We will consider different values of the upload rate $r$ in our experiments.

## 2.3 Performance Evaluation

To understand the performance of our protocol, we built a discrete-event simulator to simulate the distribution of a large file from a server to a large number of peers in the Bee protocol, so that we can comprehensively study its performance in a wide range of network configurations. For example, we may simulate a large-scale network environment with heterogeneous link capacity. Our simulator is based on the article [19]. In our simulator, the network model associates a download link and an upload link bandwidth with each peer in order to make it suitable for modeling asymmetric access networks.

However, the computational complexity of even this simple model is still complicated. Hence, we decide to simplify the network model in a higher level of abstraction way that can significantly improve the scalability of our simulation. We simplified our network model in the following ways: first, we do not consider any shared bottleneck link in the core network, so we assume the core network has infinite capacity. In addition, we also do not consider the network propagation delay and the dynamics of TCP connections. Instead, we assume that all TCP connections passing a specific link share the link bandwidth equally, so the dissemination time of a protocol is dominated by the time to transfer the file. Note that if the download capacity of receiver is the bottleneck, the spare available upload capacity of sender will be used by its other uploading connections. Finally, the endgame model [2] of BitTorrent is not simulated because it only works for a small percentage of the download time when the download time is very lengthy. This simplification of BitTorrent adversely may have a tiny impact on the dissemination time, but not a decisive one.

Obviously, there are some important network parameters that we do not simulate, such as locality properties in constructing the overlay, cross traffic impact, or malicious users. Although the above simplifications may have impact on our experiment results, we wish our simulation is able to capture some of the most important properties for the design philosophy essentially.

Table 2.1: The upload/download bandwidth distribution

| Network Type | Downloadlink | Uplink | Fraction |
|---|---|---|---|
| Homogeneous | 1500kbps | 384kbps | 100% |
| Heterogeneous | 1500kbps | 384kbps | 50% |
| | 3000kbps | 1000kbps | 50% |
| More heterogeneous | 784kbps | 128kbps | 20% |
| | 1500kbps | 384kbps | 40% |
| | 3000kbps | 1000kbps | 25% |
| | 10000kbps | 5000kbps | 15% |

### 2.3.1 Road-map of Simulations

We made our simulations to compare the dissemination time in the Bee system with the lower bound of dissemination time and the required time in the BitTorrent system. We implemented the BitTorrent system according to the detail overview of BitTorrent [2,19]. Besides, we consider three network scenarios to evaluate our protocol, each representing a different degree of heterogeneity in their upload/download capacity and the peer join pattern is set to flash crowd. Note that the lower bound of dissemination time in each scenario is calculated by Eq. 2.5 in Section 2.1. In each network scenario, we explore the impact of network size in our Bee protocol.

The bandwidth distribution of each network condition is presented in Table 2.1 First, we consider a homogeneous setting where all peers have the same upload/download capacity. Then the heterogeneous network has two types of peers, one of which has a higher upload/download capacity than the other. And then, a more heterogeneous condition with four types of peers is considered, and this network setting is the actual peer bandwidth distribution which is reported for Gnutella clients [23].

Moreover, we study the join patterns besides the flash crowd scenario. In particular we consider cases where peers with various join ratios to evaluate the impacts of join rates for the Bee and BitTorrent systems in the more heterogeneous network conditions. We also present the results of a realistic join pattern that derived from a tracker log for a Redhat 9 distribution torrent of a BitTorrent system [24].

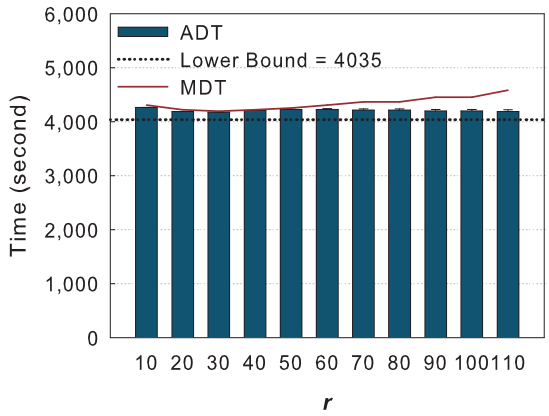


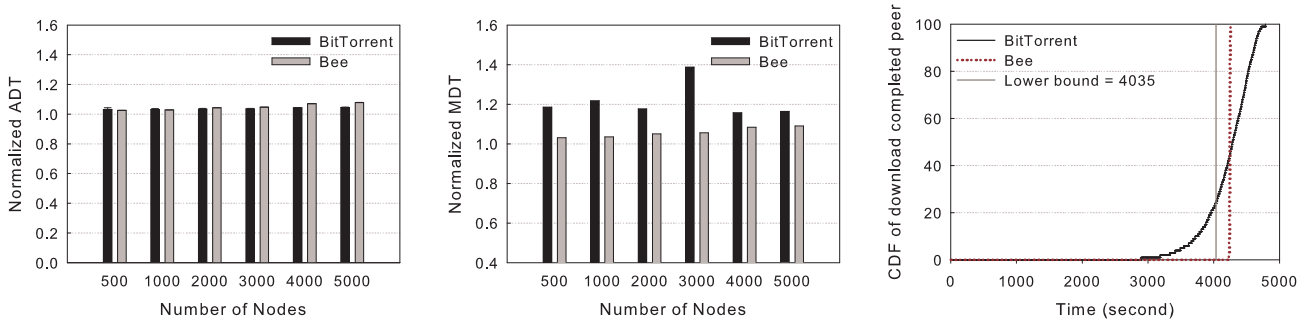

Figure 2.4: Effect of various rates $r$ (Kbps) on $ADT$ and $MDT$

Unless otherwise specified, we use the following settings in our experiments. First, we used a file size of 200MB with a block size 256KB (so a file contains 800 blocks). The seed's upload capacity is 6000Kbps. The number of contact list is 80 in both the Bee and BitTorrent system, and the maximal number of concurrent upload connections per peer is 5 in BitTorrent setting (including the optimistically unchoked connection). Then the number of initial seeds is only one in all of our experiments.

### 2.3.2 Homogeneous Environments

We start by comparing the performance of Bee with that of BitTorrent protocol in the homogeneous environment. We use the default settings as mentioned in the above description. All peers join the system at the initial stage, and leave the system when they finish their downloading. First, we study the influence of the parameter $r$ in our Bee protocol, and then evaluate the impact of network size, and upload bandwidth of seed. In this scenario, the lower bound is 4035 according to Eq. 2.5. Moreover, we do not show the upload link utilization, due to the upload link utilization of this two protocols is over 90%, which means the overall upload capacity of network is almost fully utilized.

Fig. 2.4 depicts the dissemination time (both $ADT$ and $MDT$) with $95\%$ confidence intervals from the simulations as we vary the parameter $r$ of the Bee protocol in a network with 1000 peers. In Fig. 2.4, $x$ axis shows different uploading rate $r$ (Kbps), and $y$ axis shows the $ADT$ and $MDT$. It is easy to see that when $r$ grows, the $MDT$ increases in a




(a) $ADT$ (b) $MDT$ (c)

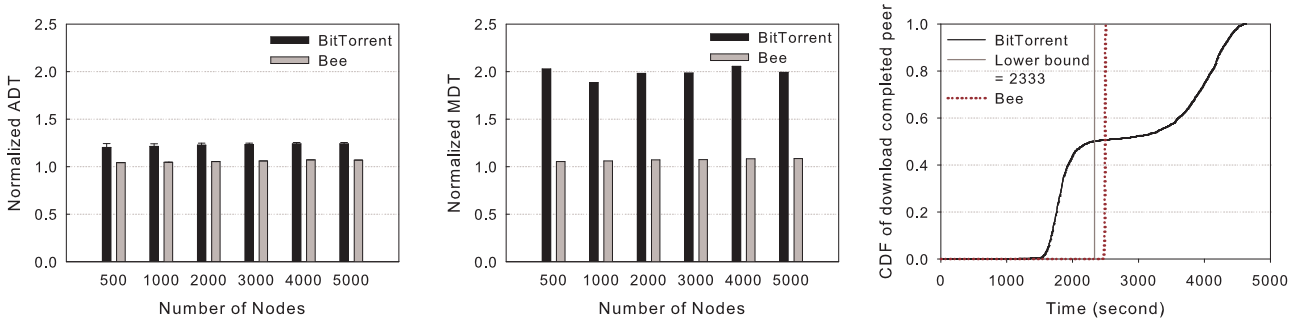
Figure 2.5: The comparison of Bee to BitTorrent in homogeneous environments

logarithmic manner and the $ADT$ of various $r$ is a smooth variation. These results show that the parameter $r$ has a little effect on the performance of Bee, and the dissemination time of Bee is close to the lower bound. Based on this experiment, we choose the uploading rate $r = 25$ Kbps in the following experiments. However, $r$ should be refined in the real Internet.

Next, we consider the impact of the scalability on the performance of Bee and BitTorrent. We use a different number of peers from 500 to 5000 in experiments, and all peers join system at the initial stage and remain in the system until they have a complete file. Fig. 2.5(a) and 2.5(b) show the normalized $ADT$ and the normalized $MDT$, respectively. The difference of $ADT$ between Bee and BitTorrent is not vary much, but the $MDT$ between Bee and BitTorrent exists a gap regardless of network size. Moreover, the difference ratio between Bee and the lower bound is only 1.1 at a network with 5000 peers. Fig. 2.5(c) shows the cumulative distribution of the number of complete peers in a network with 2000 peers. The results show that our Bee can efficiently diminish the maximum dissemination time for all network sizes, and the $MDT$ of Bee is only 4218 seconds that is very close to the lower bound.

### 2.3.3 Heterogeneous Environments

Next, we evaluate the performance of Bee and BitTorrent protocol in a heterogeneous network that consists of two types peers, one of which has a higher upload/download capacity (3000/1000 Kbps) than the other (1500/384 Kbps). As above simulations, all




(a) $ADT$ (b) $MDT$ (c)

Figure 2.6: The comparison of Bee to BitTorrent in heterogeneous environments

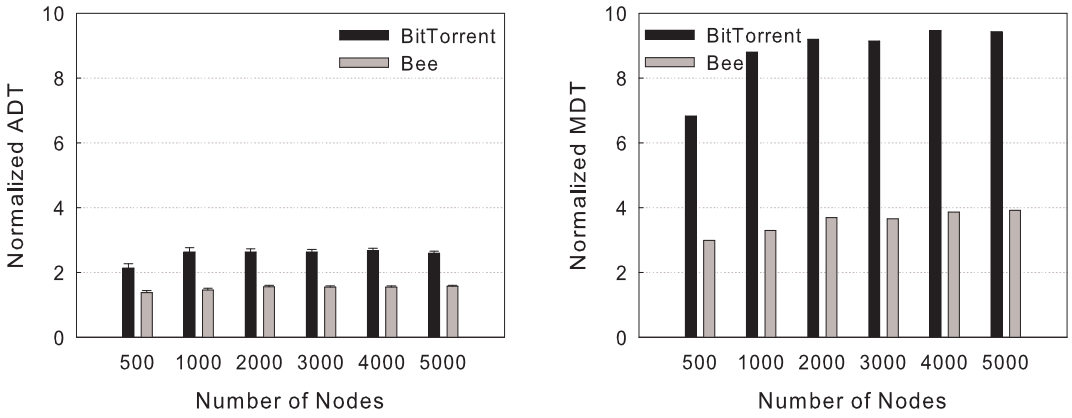
peers join the system at the initial stage, and leave when they finish downloading. In this section, we have evaluated all performance metrics of Bee but only show the impact of network size, because other results are similar to that in the homogeneous environment. In this scenario, the lower bound is 2333 seconds.

Fig. 2.6(a) shows the $ADT$ metircs for Bee and BitTorrent. It seems that both them are good on the $ADT$ metric, because their $ADT$s approach the lower bound, but our Bee is better than BitTorrent in average. Besides, Fig. 2.6(b) shows the normalized $MDT$ metric for Bee and BitTorrent, and we can see that Bee is almost twice faster than the BitTorrent in the $MDT$ metric. We also show the cumulative distribution of the number of complete peers in a network with 2000 peers in Fig. 2.6(c). The result shows that a peer with higher capacity leaves faster than the peer with lower capacity in BitTorrent. After the peers with higher capacity leave system, the system capacity deceases significantly and the dissemination time of the peers with lower capacity will be prolonged. It fits our analysis in section 2.1.

### 2.3.4 More Heterogeneous Environments

In this section, we repeated the same experiment in a more heterogeneous network with four types of peer capacity. Actually, it presents a very complex network condition. As previous experiments, we calculate the lower bound of this scenario and it should be 2089 seconds ($\frac{size(F)}{\min\{D_i\}} = \frac{1638400}{784}$).

Our first experiment is to study the impact of the number of peers from 500 to 5000.




(a) $ADT$ (b) $MDT$

Figure 2.7: The comparison of Bee to BitTorrent in more heterogeneous environments

We now explore the scalability of Bee in the complex network environment. As shown in Fig. 2.7(a), Bee has a smaller $ADT$ metric and the $ADT$ is close to the lower bound. However, BitTorrent loses the ascendency in this case, each peer in BitTorrent needs double time to finish downloading. Fig. 2.7(b) shows the normalized $MDT$ metric of Bee and BitTorrent. This result also shows that Bee is almost twice faster than the BitTorrent in $MDT$ metric. We also show the cumulative distribution of the number of complete peers in a network with 2000 peers in Fig. 2.7(a). It is easy to see that 80% peers leave system at the $\overline{T}$ time (Recall that the Eq. 2.5 that we defined in section 2.1.) and the remained 20% peers prolong the $MDT$ of Bee system. In fact, these 80% peers are higher capacity peer, and the dissemination time of remained peers is limited by their download capacity.

In order to demonstrate this phenomenon, we extend the download capacity of low-capacity peers (784 Kbps) to make sure that each peer can download a complete file before the lower bound. We increase the download capacity of the slowest peers from 784 Kbps to 1200 Kbps. Fig. 2.8(b) shows the CDFs of the download time for the two protocol in the case of a network with 2000 peers. The top one is the CDF with 784 Kbps (the minimum download capacity of some peers) and the bottom is the CDF with 1200 Kbps. From the results in Fig. 2.8(b), we can observe that the maximum dissemination time of Bee also approaches the lower bound, when the ($\frac{size(F)}{\min\{D_i\}} = 1365.3$) is smaller than ($\overline{T} = 1374.9$). The result implies that the performance of Bee is independent of the degree of network heterogeneity, and the dissemination time of Bee can approach the

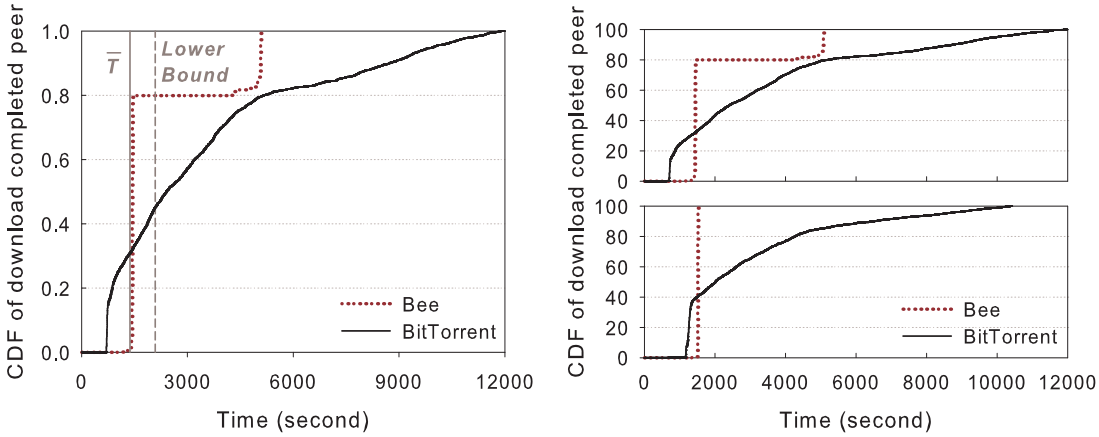


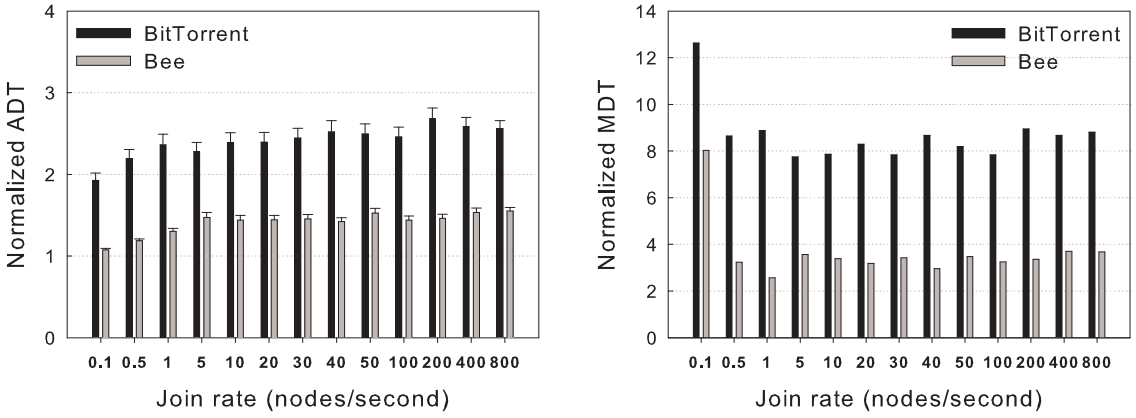

Figure 2.8: The comparison of Bee to BitTorrent without the download constraint

lower bound when there are no bottleneck links at the downstream peers. In sum, all peers in Bee are able to finish downloading very quickly even in the complex network scenario.

### 2.3.5 The Effect of Join Pattern

In this section, we study the impact of different user arrival patterns on the performance of Bee and BitTorrent systems. In following experiments, we vary the peer join rate to evaluate the performance variation of Bee and BitTorrent systems. All experiments in this section use the two following sets of settings: 1) a poisson arrival process with a total of 1000 peers. 2) an arrival pattern is derived from a tracker log of a Redhat 9 distribution torrent. And the capacity of each peer is randomly selected among four capacity types as shown in Table 2.1. The upload bandwidth of the seed is 6000 Kbps.

Fig. 2.9 shows the performance when using different user arrival rates for the Bee and BitTorrent system. As Fig. 2.9(a) shows, we can see that the $ADT$ linearly increases when the arrival rate increases both in Bee and BitTorrent. The reason for this result is that when more peers join at same time, more peers need to wait at the start-up time to receive the first block, so a larger arrival rate causes a longer $ADT$. Fig. 2.9(b) shows the $MDT$ metric of the Bee and BitTorrent. We can easily observe that when the arrival rate is low, the $MDT$s of Bee and BitTorrent both increase. The reason for this is that, when the arrival rate is low, the service capacity of overall system is also low. So the peers in




(a) $ADT$ (b) $MDT$

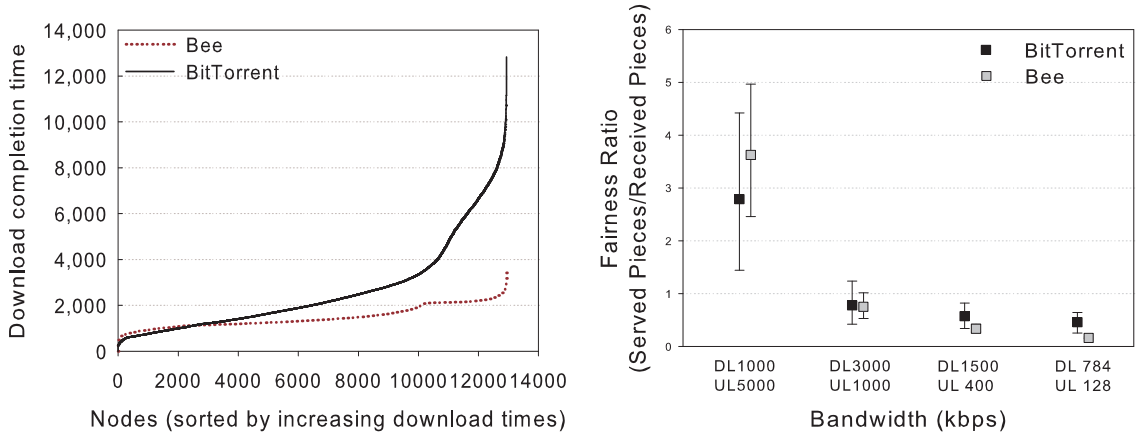
Figure 2.9: Performance comparison of Bee to BitTorrent with increasing arrival rate in more heterogeneous environment

our Bee and BitTorrent need more time to finish downloading the file. However, our Bee can outperform BitTorrent in the both metrics in such scenario.

We now evaluate Bee and BitTorrent in a realistic join pattern. In this experiment, each peer joins the system according to the tacker log of a Redhat 9 distribution torrent, the log was collected over 12,000 peers joining time. All simulation settings are the same as that in the above experiment. Note that peer capacity consists of four types as shown in Table 2.1, so the lower bound in this case is that $\frac{size(F)}{\min\{D_i\}} = 2089$ seconds.

All the results are shown in Fig. 2.10(a). First, we demonstrated the download completion time of each peer for Bee and BitTorrent in Fig. 2.10(a). It is clear that 83% peers in Bee finish the their download before 2000 seconds. On the contrary, only 50% peers can leave BitTorrent system at 2000 seconds. Moreover, Bee only needs 1/3 download time of BitTorrent to finish the file dissemination. The result shows that Bee can significantly reduce the overall download time of the file distribution system.

We especially concern how much delay occurred in the dissemination time for those peers with higher upload bandwidth. In Fig. 2.10(a), all the peers with higher upload bandwidth are the first 2,000 nodes; it is easily to see that the downloading times of the high-bandwidth peers only have a slight increase, which means the high-bandwidth peers only have a little delay in the Bee system. The result shows that our Bee protocol does **not** force the high-bandwidth peers to stay in the system.




(a) Distribution of dissemination time

(b) Fairness metrics

Figure 2.10: Performance of Bee and BitTorrent with arrival rate from Redhat 9 tracker log

In addition to the download completion time, we also concern about the fairness metric for the Bee protocol. It is clear that there are tradeoffs between fast dissemination and fairness, so we wonder how much sacrifice in fairness for the Bee protocol. The fairness metric is defined as the ratio of the amount of data sent out to that a peer received. Hence, we evaluate the fairness metrics for the Bee and BitTorrent, as shown in Fig. 2.10(b). We may see that the Bee protocol does not sacrifice too much fairness while pursuing the overall dissemination speed.

## 2.4 Related Work

In recent years, there is a tremendous interest in building content delivery networks to address the data dissemination problem, which aims to deliver a large-sized file to a group of nodes spread across a wide-area network. For the content delivery networks, several related studies, such as Bullet [4], Splitstream [5], and [25] build multiple overlay mulitcast trees or overlay meshes to establish a more efficient system to deliver data in the Internet. Though creating and maintaining multicast trees among peers provides an alternative solution for content delivery, however, it may incur high maintenance cost due to the characteristics of churn effect in a P2P network.

The popular content distribution system, BitTorrent [2] is successful for its efficiency

in delivering a large file. There are two key mechanisms used in the BitTorrent system, namely, the "Tit-For-Tat" (TFT) peer selection policy and the local rarest first piece selection (LRF) strategy. Many varieties of BitTorrent are proposed to improve its performance. Wu *et al.* [26] try to reduce the download time of overall BitTorrent peers for flash crowd arrival pattern. In additional, Slurpie [3] focuses on reducing load on servers and clients download times when the number of downloading clients is huge. Crew [6] is a new gossip-based protocol for data dissemination, and it performs a faster dissemination than the BitTorrent protocol in experiments, and [27] is another similar work. Compared with the above efforts, our work makes a difference contribution on minimizing data dissemination time.

Content Delivery Networks (CDN), such as Akamai Technologies [28], have been proposed to improve accessibility for the commercial companies. CDNs are dedicated collections of servers located strategically across the wide-area Internet. Content providers, such as multimedia video sources, contract with commercial CDNs to distribute content. CDNs are compelling to content providers because the responsibility for distributing content is offloaded to the CDN infrastructure. Many new infrastructures for the CDNs have recently been developed to focus on distributing large files, such as CoBlitz [29] system. These systems offer a stable and performance predictable content delivery architecture, especially for the businesses that want to offload their bandwidth but need to delivery a large content. However, regardless of how many nodes in the CDNs are deployed, in case number of users grows too fast, performance of these systems may degrade significantly.

## 2.5 Conclusion

In this chapter, we present our experimental study of the data dissemination problem. We also present the Bee protocol, which includes a slowest peer first strategy and a topology adaptation algorithm to maximize the speed of dissemination. Under the slowest peer first strategy, a peer always transmits blocks to the neighbors that have fewest number of downloaded blocks. Bee protocol also embeds a topology adaptation algorithm for a peer to adapt number of connections to its neighbors based on its own upload bandwidth. Moreover, our experimental results show that the dissemination time of the Bee protocol

approaches lower bound of the data dissemination problem for both homogenous and heterogeneous network environments. Specifically, in the simulations on heterogeneous network environment presented in this chapter, dissemination time of Bee is only 1/5 of that of BitTorrent, which is only 10% higher than the theoretical lower bound. As for the arrival traffic derived from a software release log, dissemination time of Bee is only 1/3 of that of BitTorrent system. Besides, we also show that the Bee protocol does not sacrifice too much fairness while pursuing the dissemination speed, and those peers with higher upload bandwidth only have a little delay in the dissemination time. We do not devised with a mechanism to detect free riders, this is an interesting emerging research issues and will be investigated further.

# CHAPTER 3

# Efficient Big Data Broadcasting: An Algorithmic Approach

"*We reject: kings, presidents and voting. We believe in: rough consensus and running code*."

–David D. Clark

BIG-Data Computing, a new critical challenge has sparked the major research issues that are reshaping ICT industry and scientific computing in the past few years [8]. The rapid advances in ICT technologies, such as computation, communication and storage have resulted in enormous data sets for business, science and society being generated and analyzed to explore the value of those data.. Currently, both ICT industry engineers and scientific researchers may have to deal with petabytes of data sets in the cloud computing paradigm [9]. For instance, in industry, Google, Yahoo!, and Amazon collect huge data every day for providing the information freely to people in useful ways. In science, the Large Hadron Collider (LHC) can generate about fifteen petabytes of data annually, and thousands of scientists around the world need to access and analyze this big data sets [30]. Thus the demand for building a distributed service stack to efficiently distribute, manage and to process massive data sets has risen drastically.

In the past decade, several efficient techniques are proposed to manipulate from terabytes to petabytes of data and run on distributed manners consisting of as many as tens of thousands of machines. For example, Google presented a software framework, named MapReduce [31] to process a large-scale data distributively, and also proposed Bigtable [32] for storing the structured data on thousands of machines. The success key of these techniques is to adopt a data-parallelism model with general communications [33]. Data-parallelism can be defined as a computation model applied independently to each data item of a set of data which allows the degree of parallelism to be scaled with the

volume of data[1]. There are many of significant issues for developing data-parallelism applications, such as, the effective strategy for data decomposition, load balancing on processing nodes, and data transmissions between a large set of nodes [10]. In particular, for big-data computing, data transmission operations may become one of most significant issues to reduce the job completion times. For instance, Facebook shows that data transmission time occupies approximately one-third running time of jobs in its Hadoop tracing logs [34].

In this chapter, we focus on the big data broadcasting operation that is one of most essential and fundamental communication mechanisms in many distributed systems. There are a lot of application domains that widely apply broadcasting operations, such as scientific data distributions [35], database transaction backups, the latest security patches, multimedia streaming applications, and data replica or virtual appliance deployment [36,37] among distributed data centers. Since the size of data becomes more and more enormous, the significance of broadcasting operation becomes important increasingly.

We discuss the Big data broadcasting problem both in practical and theoretical ways. We consider a heterogeneous network system with nodes, *i.e.*, nodes may have different uploading capacities. The Big data broadcasting problem is about how to let these nodes efficiently obtain the entire bulk data cooperatively, and how to minimize the total transmission time in each node. Here we focus on the problem of how to build a single optimal tree to broadcast multiple chunks from a single source. We assume that there are $n$ nodes in a heterogeneous network system, denoted by $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \ldots, \mathcal{N}_n$, where the broadcasting source is node $\mathcal{N}_1$ and the $n$ nodes have upload capacities $c_1, c_2, c_3, \ldots, c_n$, measured in kilobyte per second (KBps). We also assume that the source has the data item that is divided into $m$ chunks of equal size, to disseminate to all the other nodes. We also assume that the downloading capacity of each node is larger than or equal to its uploading capacity.

We introduce a novel LockStep Broadcast Tree (LSBT) to model the original Big Data broadcast problem [38,39]. Then we present a polynomial-time algorithm to build an optimal LSBT, that is a broadcast tree where data chunks can be sent in a pipelined fashion with a good throughput. The main idea in our LSBT model is to define a basic unit of

[1] http://en.wikipedia.org/wiki/Data_Intensive_Computing#Data-Parallelism

upload bandwidth, $r$, such that each node uses only integer multiples of $r$ in broadcasting. In so doing, the number of upload connections is proportion to the capacity of a node. Furthermore, we also divide the broadcast data into $m$ chunks. These data chunks are then broadcast down the tree by the nodes in a pipeline manner. We show that based on the LSBT model, the maximum number of rounds required to complete the broadcast of entire data chunks is O$(m + logn)$ steps, where $n$ is the number of nodes. In a homogeneous network environment in which each node has the same uploading capacity $c$, we show that the optimal uplink rate $r^*$ of LSBT is is either $c/2$ or $c/3$. For heterogeneous networks, we present an O$(n \log^2 n)$ algorithm to select the optimal uplink rate $r^*$ and to construct the optimal LSBT. Numerical results show that the maximum completion time of our LSBT approximates to the optimum of the big data broadcast problem.

Our main contributions are as follows.

1) Propose the LockStep Broadcast Tree (LSBT), which models the original big-data broadcasting problem to support heterogenous networks by constructing an efficient pipeline broadcasting tree. We contribute to the understanding and investigation of how to design a scalable and practical algorithm that minimize the maximum completion time of a single tree solution via LSBT model. To the best of our knowledge, little work has been conducted on the design of the heuristics algorithm for the data broadcasting problem.

2) Design a novel polynomial-time algorithm to select the optimal uplink rate of $r^*$ for building an optimal LSBT. Unlike the previous works [40,41] in which the criteria is to maximize system throughput, to the best of our knowledge, this is the first to study the problem of designing a tree overlay network aiming at minimizing the maximum completion time.

3) Introduce several original applications based on the LSBT model. Specifically, given a data delivery deadline, LSBT may be possible to determine that can the network system meet the deadline or what is the possible deadline for the delivery job. This advantage can maximize the performance of collaborative applications in datacenter networks.

The rest of this chapter is organized as follows. In the Section 3.1, we give background of the Big data broadcasting problem. The Section 3.5 describes the context of related work. We state the general Big data broadcasting problem and introduce our LSBT model and its applications in the Section 3.2. The detail of our optimal algorithm

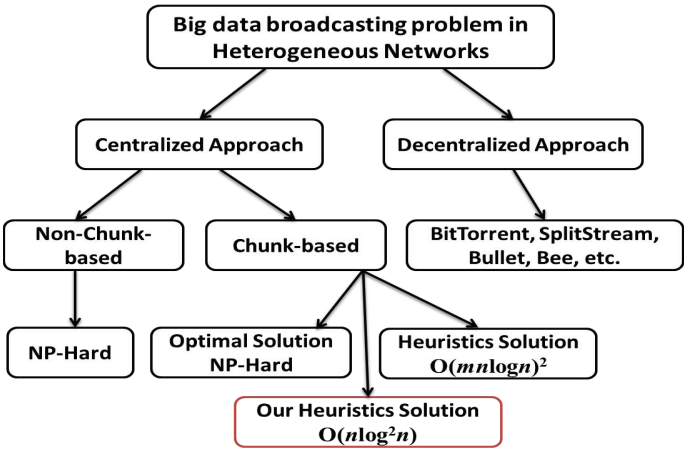



Figure 3.1: Scope of our contributions

for LSBT problem is presented in Section 3.3. Numerical evaluation are presented in Section 3.4. We conclude this chapter in Section 3.6.

## 3.1 Background

Supposed that $m$ data chunks of equal size are initially held by a single source node in a network. The data broadcasting problem is about disseminating these $m$ chunks to a population of $n$ nodes in as less time as possible, subject to the uploading link capacity constraints of nodes. This problem has been studied in the context of many different network scenarios, such as homogenous and heterogenous networks . For interested readers, a comprehensive survey can be found in the article [42]. In this chapter we focus on big data broadcasting problem in heterogenous networks. Figure 3.1 illustrates these solutions to the big data broadcasting problem in heterogenous networks along multiple dimensions.

For the centralized approach, we first look at the results of the Non-Chunk based approach. Khuller and Kim [38] showed that the problem of minimizing the completion time for broadcasting a single chunk (a message) in heterogenous networks in a NP-hard problem. The authors also showed the Fastest-Node-First (FNF) heuristic method gets a performance ratio of at most 1.5 and the FNF results in optimal solutions in many cases for single chunk broadcast. In additional, Liu [43] showed that the FNF heuristic method is optimal in only two classes of nodes. However, the data broadcasting problem is more

complicated when the data consists of multiple chunks and it is still an open problem: Can data broadcasting problem with multiple chunks be solved by a polynomial time algorithm? [44].

Within the Chunk-based methods, the optimal solution has been shown in the article [39]. The authors presented an uplink-sharing model for the well-known data broadcasting problem and formulated data broadcasting problem as a mixed integer linear programming (MILP). However, as the numbers of variables in the linear programming grows exponentially $n$ and $m$, this method is not practical for large $n$ and $m$. Goetzmann *et. al.* [45] show that if peer capacities are heterogeneous and symmetric, this problem becomes strongly NP-hard. A recent result [46] presented two heuristic algorithms to schedule data chunks transfer between nodes. The time complexity of both two centralized algorithms is O$(m \times nlogn)^2$.

For the decentralized approach, many decentralized systems have been proposed to disseminate chunks via an overlay topology. With overlay-based approaches, nodes maintain a set of overlay links to other nodes and exchange chunks among neighboring nodes. BitTorrent [2], SplitSteam [5], Bullet [4] and Bee [47] are some examples of the overlay-based approach. In [47], the authors showed Bee can approach lower bound of the maximum completion time in heterogenous networks by simulations. In this chapter, we retain the interest in the centralized approaches, thus interested readers can find a comprehensive survey of these decentralized systems in the article [48].

## 3.2 Problem Statement

The assumption in our model is similar to the Uplink-Sharing model proposed by J. Mundinger *et al.* [39]. Each node can simultaneously connect to other nodes and the available upload capacity of a link is shared equally amongst the uploading connections. Based on the Uplink-Sharing model, we model the nodes and data transfer networks as the nodes and edges of a direct graph. We assume that there are $n$ nodes in a network system, denoted by $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$,. . . , $\mathcal{N}_n$, where the broadcasting source is node $\mathcal{N}_1$ and the $n-1$ nodes have upload capacities $c = \{c_1, c_2, c_3,\ldots, c_n\}$, measured in kilobyte per second (KBps). Besides, we also assume that the source node, $\mathcal{N}_1$, has the data item

that is divided into $m$ chunks of equal size, to disseminate to all the other nodes, and $c_1$ is larger than or equal to that of other nodes. Finally, we assume that the downloading capacity of each node is larger or equal to its uploading capacity. This is true for virtually all existing network access technologies, e.g., ADSL or cable modems.

### 3.2.1 LockStep Broadcast Tree (LSBT) problem

To reduce the complexity of the original data broadcasting problem [38,39], we model it as the LockStep Broadcast Tree (LSBT) problem. By this we define a performance goal for a single LSBT, that is achieving minimum completion time by optimizing the basic bandwidth allocation, $r$, among LSBT nodes. Different from original problem, we allow data be divided into chunks and sent in a pipeline fashion. Formally, given a set of $n$ nodes $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$, each node $\mathcal{N}_i$ is connected to the network via an access link of upload capacities $c_i$ and a size of chunks $B$. The LSBT problem is to determine the upload bandwidth $r^*$ of each uplink to build the LSBT $t$, in which node $\mathcal{N}_i$ should allocate upload bandwidth $r^*$ to each connection to its child nodes in order to minimize the maximum completion time $D$ for propagating a data chunk. Note that it is possible to handle simultaneously several connections and to fix the bandwidth allocated to each connection [49]. In the following definition, we define the number of edges $k$ in each node for LSBT.

**DEFINITION 3.2.1** *For each LSBT node $\mathcal{N}_i$, the number of edges (uploading connections) $k$ is depended on its upload capacity, i.e., $k = \lfloor \frac{c_i}{r} \rfloor$, for $1 \leq i \leq n$ and $\forall\, r \in \mathbb{R}^+$.*

The formal mathematic definition of the maximum completion time $D$ is shown as follows.

$$r^* = \arg\min_{r \in \mathbb{R}^+} D(c, r) = \arg\min_{r \in \mathbb{R}^+} \sum_{i=1}^{h(t^{(c,r)})} \frac{B}{r}, \tag{3.1}$$

where $t^{(c,r)}$ is the LSBT with the set of upload capacity $c$ and an upload bandwidth $r$, $h(t^{(c,r)})$ describes the function that returns the height of the LSBT $t^{(c,r)}$.

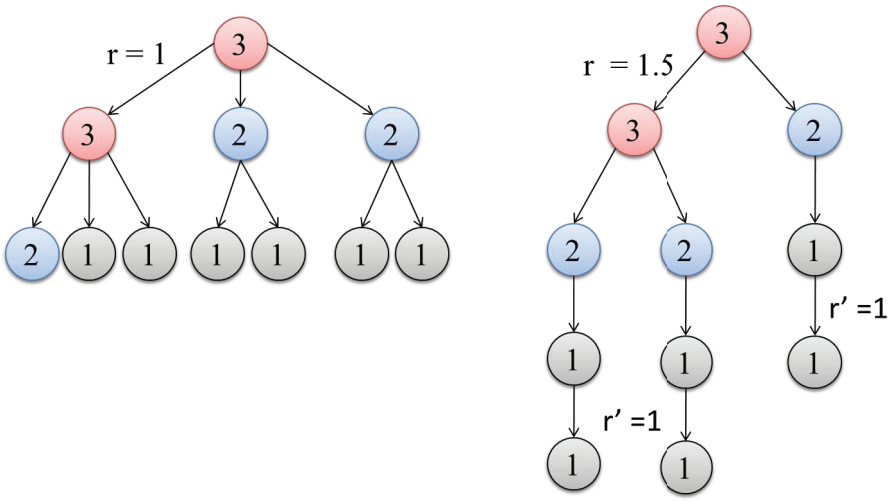



Figure 3.2: The two examples of LSBT. The tree (a) presents the optimal LSBT with $r^*$ = 1, the maximum completion time $D$ is 2 units of time and tree (b) requires 3 units of time. Assume that the size of data chunk $B = 1$ and the digits specify these node's upload capacity.

Note that this general Equation (3.1) removes restrictions on the location of nodes in the network, it only calculates the propagation delay of data chunks from the root to the leaves. Moreover, LSBT model addresses the data broadcasting problem by building a single broadcast tree, in which nodes can transmit data chunks in a pipeline manner. Thus the maximum completion time $D$ is the summation of the transmission time of a data chunk (*i.e.*, $\frac{B}{r}$) in each level of the LSBT $t^{(c,r)}$.

**Example**. Figure 3.2 shows the two examples. Given a set of eleven nodes having upload capacities {3, 3, 2, 2, 2, 1, 1, 1, 1, 1, 1}, and we can build at most $11^{11-2}$ different broadcasting trees (by Cayley's formula [50]). However, there exists an optimal LSBT that constructed by in order of decreasing edges of nodes. We will show this important property of LSBT in the next section (**THEOREM** 3.3.3). In Figure 3.2, the tree (a) presents the optimal LSBT $t^a$ with the optimal upload bandwidth $r^* = 1$. Here, we assume that the size of data chunk $B = 1$. Since $h(t^a) = 2$, the maximum completion time $D$ of tree $t^a$ should be 2 (*i.e.*, $\frac{1}{1} + \frac{1}{1} = 2$) units of time (by Equation 3.1). The other tree $t^b$ in Figure 3.2 is not an optimal LSBT. The maximum completion time $D$ of tree $t^b$ is 3 units of time (*i.e.*, $\frac{1}{1.5} + \frac{1}{1.5} + \frac{1}{1.5} + 1 = 3$). Note that in tree $t^b$ these gray nodes in

the $3^{th}$ level only can provide one unit of upload capacity to their child nodes even if $r$ is specified as $1.5$.

### 3.2.2 Potential Applications of LSBT

We envision that our LSBT could be well-suited for a host of applications. There are at least three broad applications where LSBT can be applied: 1) topology control in BitTorrent-like systems; 2) data broadcasting in cloud computing software stack; 3) energy conservation in peer-assisted content delivery services. We consider these in the context of network systems that are heterogenous network environments. First, Our algorithm of LSBT could be useful in topology control in BitTorrent-like systems [2]. BitTorrent is a peer-to-peer application that aims to enable the fast and efficient distribution of large files among a large group of nodes. In BitTorrent, each peer maintains a constant number of concurrent upload connections (usually five). Please see the article [2] for more detailed descriptions. Recent studies [19,47,51] show that the fixed upload connections limit is harmful to uplink utilization and peer fairness in BitTorrent. However, how to decide an appropriate number of concurrent uploads in BitTorrent still is a challenge. The proposed algorithm for LSBT may provide an insight into selecting the number of concurrent uploads in BitTorrent-like systems.

Second, LSBT can be integrated into the cloud computing software stack. For example, Apache Hadoop[2] is a software framework that allows for the distributed processing of large data sets across clusters with thousands machines. Thus an efficient and scalable way to disseminate a large volume of data among machines is a significant challenge in Hadoop [34]. Another example is the delivery services of OpenStack[3], it is designed for virtual appliance deployment in datacenters. Our LSBT can be integrated into the delivery services of OpenStack software stack. A number of algorithms and protocols have been proposed, implemented, and studied [34,52]. For any data delivery job initiated by cloud computing softwares, there is an associated deadline. The main advantage of LSBT is to enable these cloud software stacks to predict and schedule the associated deadline of a data delivery job. Specifically, given a data delivery deadline, LSBT may be possible

[2]http://hadoop.apache.org/
[3]http://openstack.org/

to determine that can the network system meet the deadline or what is the possible deadline for the delivery job. This advantage can severely impact application performance in datacenter networks.

Finally, Our algorithm for LSBT could be useful to answer the question: what is the maximum streaming rate that can be sustained for all receivers within a peer-assisted content delivery service provider. Many content delivery service providers, such as PPLive[4], Akamai[5], that may rely on participating users contributing uplink bandwidth to scale up delivery services to hundreds of thousands of users. However, if the total contributed bandwidth from the service provider and participating users can not support to the demanded quality of services (*ex.*, H.264[6]/768kbps), the service provider should increase contributed bandwidth (servers) from server-side. For energy conservation and environmental issues, it is an interesting and significant issue to investigate how to dynamically increase or decrease the number of servers in accordance with the demanded QoS and the number of active users. Given a set of node upload capacities $c$, our LSBT algorithm can roughly sketch out the coarse-grained QoS level (*i.e.*, $r^*$ Kbps) of the current system and be used to regulate the energy consumption in server-side. Thus, our LSBT model can use for creating a systematic approach that arranges server-side resources for peer-assisted content delivery protocols. To the best of our knowledge, little work [53] has been conducted on energy conservation in peer-assisted content delivery services. In future work, we aim to apply our LSBT algorithm to the research direction.

## 3.3 Optimal LockStep Broadcast Tree

In this section, we present our LSBT algorithm that is also a heuristic for the data broadcasting problem. Given a set of node upload capacities $c$, we aim at finding an optimal LSBT, that is a data broadcast tree where data chunks can be sent in a pipelined manner. We provide a thorough analysis of LSBT in both homogenous and heterogenous network systems. We first clarify LSBT in homogenous networks cases and describe the LSBT

[4]http://www.pptv.com/
[5]http://www.akamai.com/
[6]http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC

algorithm in heterogenous network cases later.

### 3.3.1 Homogenous Network Systems

We present the optimal solution of LSBT when the upload capacities of nodes are identical, *i.e*., homogeneous. In general, we assume that all nodes have upload capacity of $c$. Mundinger *et al*. [39] have presented the optimal scheduling solution for broadcasting multiple messages. The following **THEOREM** 3.3.1 is proved in the article [39]. If each round costs one units of time, then the maximum completion time of the optimal solution is $m + \lfloor \log_2 n \rfloor$, where $m$ is the number of chunks and $n$ the number of nodes. Note that each node can only upload one data chunk to another node in each round. By contrast, each node can send a data chunk to $k$ other nodes simultaneously in the LSBT model.

**THEOREM 3.3.1** *In homogenous network systems, the minimum number of rounds required to complete the broadcasting of all data chunks is* $m + \lfloor \log_2 n \rfloor$*, where* $m$ *is the number of data chunks and* $n$ *is the number of nodes.*

In our LSBT model, the maximum completion time $D$ is equal to Equation (3.1). However, due to the upload capacities of all nodes are equal, it can be simply expressed as follows (note that $r = \frac{c}{k}$).

$$\begin{aligned} D &= \frac{B}{r} \log_k n \\ &= \frac{kB}{c} \frac{\ln n}{\ln k}, \end{aligned}$$

where $B$ is a size of data chunks.
Let $\mathcal{G} = \frac{B \ln n}{c}$, we have

$$D = \mathcal{G} \frac{k}{\ln k}. \tag{3.2}$$

Set $\frac{dD}{dk} = 0$,

$$\frac{dD}{dk} = \frac{\mathcal{G}}{\ln k} - \frac{\mathcal{G}}{(\ln k)^2} = 0. \tag{3.3}$$

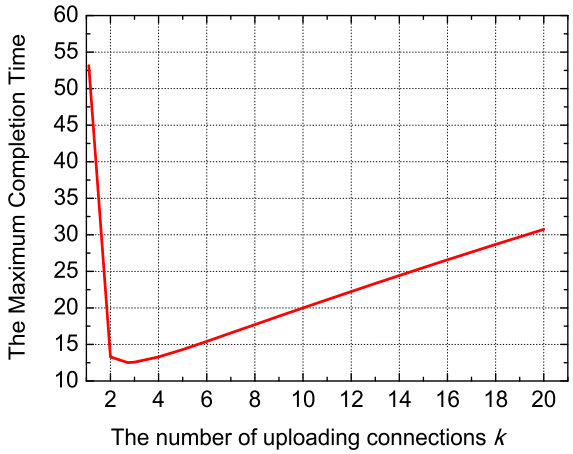



Figure 3.3: Numerical results of LSBT in homogenous networks. Assume that the number of nodes $n = 100$, the size of data chunk $B = 1$, and the upload capacity of all nodes $c = 1$.

The Equation (3.3) implies that

$$\ln k = 1,$$

$$k = e.$$

It can be shown that Equ. 3.2 is a convex function. Thus we have the following theorem in discrete model.

**THEOREM 3.3.2** *In homogenous network systems, the optimal value $r^*$ for LSBT is is either $c/2$ or $c/3$ that makes the LSBT minimizes the maximum completion time, where $c$ is the upload capacity of all nodes.*

Figure 3.3 illustrates a simple numerical example of LSBT in a homogenous network, in which we set $n = 100$, $c = 1$, and $B = 1$. We then calculated the maximum completion time $D$ in Equation (3.2). In the results, all nodes have $k$ upload connections, the value of $k$ depending on the considered scenario. We can see that the numerical results significantly depend on the value of $k$ in homogenous network systems, and the LSBT can minimize the maximum completion time when $k$ is equal to $e$ as we shown in **THEOREM** 3.3.2.

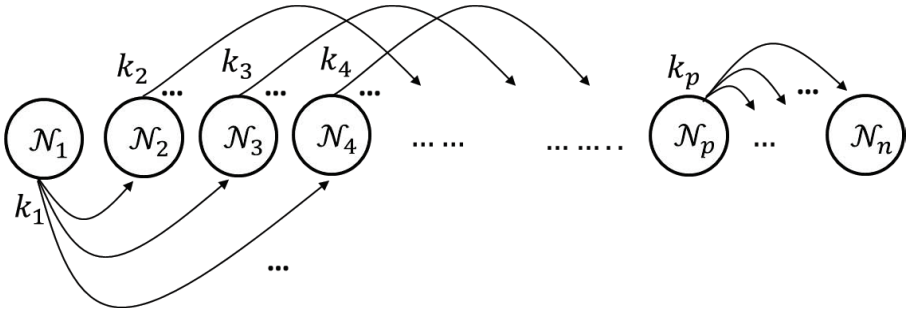



Figure 3.4: An illustration of building a LSBT

### 3.3.2 Heterogenous Network Systems

We now consider the general LSTB model in which nodes' upload capacities may be different. First, we present an algorithm to construct an optimal LSBT for a given rate $r$. We then give both the upper and lower bounds of the value of $r^*$. Finally we present an O$(n \log^2 n)$ algorithm to select the optimal upload bandwidth $r^*$ of each uplink and to construct the optimal LSBT.

We now present the algorithm GLSBT to construct an LSBT $t$ which is shown to be optimal for the given rate $r$. Given a set of nodes $\mathcal{N} = \{n_1, n_2, \cdots, n_n\}$ with $c_i$ as the upload capacity of node $i, 1 \leq i \leq n$, and a real number $r$ to denote the rate of the LSBT. We assume that the nodes are given in non-increasing order of their upload capacity, *i.e.*, $c_j \leq c_i$ if $i < j$.

**Algorithm GLSBT**$(c, r)$

**BEGIN**

1) Given $r$ as the rate of the LSBT, the degree of each node $n_i$ is given by $k_i = \lfloor c_i/r \rfloor$.

2) Construct the LSBT $t$ by assigning a node $n_q$ as the parent of the node $n_l$ if and only if $q$ is the smallest integer such that $l \leq 1 + \sum_{i=1}^{q} k_i$, where $1 \leq l \leq n$.

**END**

The following theorem shows that the algorithm GLSBT gives an optimal LSBT for the given rate $r$.

**THEOREM 3.3.3** *Given an uplink rate $r^*$, building the LSBT $t$ that is constructed in a way that any child node's out-degree is always less than or equal to its parent's and providing that $t$ is optimal in terms of the maximum completion time $D$.*

**Proof**: Suppose that we have a set of $n$ nodes, and $k_1, k_2, k_3, \ldots, k_n$ are the edges of

nodes for building a LSBT. Then by **DEFINITION** 3.2.1: $k_i = \lfloor \frac{c_i}{r} \rfloor$, for $1 \leq i \leq n$. Let $k_1, k_2, k_3, \ldots, k_p$ be the edges of nodes in a LSBT as shown as Figure 3.4, where $p$ is the largest integer, and $\sum_1^p k_p \geq (n-1)$. There are two cases impact on the height of LSBT.

Case I) Assume that an optimal LSBT $t$ is created by the order $(k_1, k_2, k_i, \ldots, k_j, k_p)$ and $k_i < k_j$, for $1 \leq i < j \leq p$. Let $h$ be the height of $t$. If $k_i$ and $k_j$ are interchanged, then there exists another optimal LSBT $t'$ with height $h'$ in order by $(k_1, k_2, k_j, \ldots, k_i, k_p)$. Since $k_j > k_i$, $h'$ is less than or equal to $h$. Therefore $t'$ is an optimal LSBT (*i.e.*, $\frac{Bh'}{r^*} \leq \frac{Bh}{r^*}$).

Case II) Assume that an optimal LSBT $t$ is created by the order $(k_1, k_2, k_i, \ldots, k_p, k_j)$ and $k_i < k_j$, for $1 \leq i < j \leq n$. Similarly in switching $k_j$ and $k_i$, we get the new LSBT $t'$. Since $k_j > k_i$, the height of $t'$ is also less than or equal to the height of $t$. Therefore $t'$ is an optimal LSBT. □

Then we provide lower bound and upper bound of the value of $r^*$ as follows.

**LEMMA 3.3.1** *(Lower bound) In heterogenous network systems, the lower bound of $r^*$ in the optimal LSBT is larger than or equal to $\frac{c_1}{n-1}$, where $c_1 \geq c_i$ for $1 < i \leq n$.*

**Proof**: If this is not true (*i.e.*, $r^* < \frac{c_1}{n-1}$), then we have the optimal LSBT $t'$ where $r' < \frac{c_1}{n-1}$. There exists another LSBT $t$, where $h(t) = 1$. The value of $r^*$ in $t$ is equal to $\frac{c_1}{n-1}$ and the value of $D$ in $t$ is equal to $\frac{B(n-1)}{c_1}$. However, the value of $D$ in $t'$ is larger than $\frac{B(n-1)}{c_1}$. This contradicts the assumption that $r^* < \frac{c_1}{n-1}$. □

**LEMMA 3.3.2** *(Upper bound) In heterogenous network systems, the upper bound of $r^*$ in the optimal LSBT is less than $\frac{\sum c_i}{n-1}$, for $1 \leq i \leq n$.*

**Proof**: A tree has $n$ vertices and $n-1$ edges. It implies

$$r^* \times (n-1) \leq \sum_{i=1}^{n} c_i.$$

However, the leaf nodes in LSBT can not contributes their upload capacities, thus

$$r^* \leq \frac{\sum_{i=1}^{(n-l)} c_i}{n-1} < \frac{\sum_{i=1}^{n} c_i}{n-1},$$

where $l$ is the number of the leaf nodes in a LSBT. □

Next, we give the details of the algorithm for the selection of $r^*$. As described in Equation 3.1, $r \in \mathbb{R}^+$, so it means the possible value of $r$ is infinite, even both the upper and lower bounds of the value of $r^*$ are given. Since the number of $r$ is infinite, an efficient discretization algorithm of $r$ is critical. In LSBT, we propose a simple division algorithm to discretize the value of $r$. This algorithm comes from the observation: the number of upload connection (*i.e.*, $k$) in each LSBT node is a positive integer and $1 \leq k \leq (n-1)$. Thus we enumerate all possible candidates of $r^*$ which make $k$ an integer. Algorithm 3.1 presents our solution to discretize the value of $r$. Let $CandidateSet$ denote the set of the possible value of $r^*$, and the binary search will be performed on it.

In Algorithm 3.1, it first reduces the redundance of $c_i$ by preforming an union operation (named $UnionSet$) of each $c_i$, for $1 \leq i \leq n$ and sorting the set (in line 4-7). Next, the loop from line 8 to 18 is used to discretize the value of $r$ and to filter out the extreme $r$ values restricted by the upper and lower bounds. In the loop, it gets candidates of $r$ by computing $u/k, \forall\, u \in UnionSet$ and $1 \leq k \leq (n-1)$, and puts those candidates into the $CandidateSet$. Note that the number of candidates is O($n^2$) if each LSBT node has an unique upload capacity. However, the filter scheme can significantly reduce the number of candidates. We will show the experimental results in the next section.

Before we present the binary search algorithm for selecting the value of $r^*$, we first show the following lemma and theorem which provide properties to derive the efficient binary search algorithm on $r^*$.

**LEMMA 3.3.3** *Given the discrete spectrum of $r$ for building a LSBT $t$, the value of $r^*$ occurs in one of the values that change the height of $t$.*

**Proof**: Suppose that the lemma is not true, there exists an optimal LBST $\hat{t}$ built with the value of $\hat{r}$, its height is $h$, and the next value of $\hat{r}$ in the discrete spectrum (labeled as $r'$ and $r' = \hat{r} + \delta,\ \delta > 0$) does not increase the height of $\hat{t}$. According to Equation (3.1) and $r' > \hat{r}$, there is another LSBT $t'$, its height is $h$, and the maximum completion time of $t'$ is less than the one of $\hat{t}$. This contradicts the assumption that $\hat{t}$ is an optimal LSBT of height $h$. □

**Algorithm 3.1** A discretization algorithm for the candidateset

**Input:** a set of upload capacities $c$ and the ***upper*** and ***lower*** bounds of $r^*$
**Output:** $CandidateSet$

```
1: BEGIN
2: UnionSet ← empty
3: CandidateSet ← empty
4: for i ← 1 to n do
5:    UnionSet ← UnionSet ∪ c_i
6: end for
7: UnionSet ← Sort(UnionSet)
8: for k ← 1 to n − 1 do
9:    for all u in UnionSet do
10:      r ← u/k
11:      if r ≥ upper then
12:         continue
13:      else if r < lower then
14:         break
15:      end if
16:      CandidateSet ← CandidateSet ∪ r
17:   end for
18: end for
19: return CandidateSet
20: END
```

**THEOREM 3.3.4** *There exists an optimal LSBT in which the height of this optimal LSBT is bounded by* $\lfloor \log_2 n \rfloor + 1$*, where* $n$ *is the number of nodes.*

**Proof**: Let $\mathbb{T}$ denote the set of optimal LSBT. Suppose that the lemma is not true, then $h(t) > \lfloor \log_2 n \rfloor + 1$, $\forall t_i \in \mathbb{T}$. Then considering $t_x$ as the tree with the largest uplink rate in the set of optimal LSBT $\mathbb{T}$, $r_x$ denotes the uplink rate of $t_x$. Since $t_x$ is one of the optimal LSBT, its maximum completion time $D_x$ is optimal and $D_x = h(t_x)\frac{1}{r_x}$.

Let $r_y = \frac{r_x}{2}$, then $h(t_y) \leq \frac{h(t_x)}{2}$. Since $r_y = \frac{r_x}{2}$, each node's degree in $t_y$ is double of that in $t_x$ and the height of $t_y$ is at least half of $t_x$. Besides, doubled degree of a tree creates logarithmic decreasing in the height of the tree, so that $h(t_y) \leq \lfloor \log_2 n \rfloor + 1$. Then

the maximum completion time of $t_y$ is as follows,

$$\begin{aligned} D_y &= h(t_y)\frac{1}{r_y} \\ &= h(t_y)\frac{2}{r_x} \\ &\leq \frac{h(t_x)}{2} \times \frac{2}{r_x} \\ &= D_x, \end{aligned}$$

from which we can conclude that $t_y$ is an optimal LSBT. It contradicts the assumption so there must be a tree in the set optimal LSBT $\mathbb{T}$ and its height $h \leq \lfloor \log_2 n \rfloor + 1$. □

Algorithm 3.2 describes our scheme to search the value of $r^*$ to build an optimal LSBT. Searching the $r^*$ is much like searching a binary search tree, except that instead of searching the value of $r$, it make a seeking condition both on the value of $r$ and the height of LSBT $h$. Algorithm 3.2 takes as input a set of upload capacities $c$ and $CandidateSet$ obtained by Algorithm 3.1. For each different height of LSBT (in line 5-27), it searches the optimal value of $r^*$ and returned the best $r^*$ and the maximum completion time (in line 28). The value of $h$ is restricted to $\lceil log_2 n \rceil + 1$ (in line 5) because of **THEOREM** 3.3.4. Thus we can only check the height of LSBT from 1 to $\lfloor \log_2 n \rfloor + 1$. During the loop (in line 8-26), Algorithm 3.2 performs a straightforward generalization of the binary searching procedure. In line 11, $GLSBT()$ is an O($n$) function for building a LSBT according to a specified $r$ and it returns the LSBT $t$. Lines 13-19 check to see if we have now discovered the value of $r^*$ for the specified $h$, and update the best $r^*$ and $D^*$ if we have. Note that by **LEMMA** 3.3.3, the line 12 presents the successful condition for searching. The line 19 terminates the search unsuccessfully, *i.e.*, an optimal LSBT of height $h$ does not exist. Based on Algorithm 3.2, we can easily prove the following theorem (the detailed proof is omitted due to space limitation).

**THEOREM 3.3.5** *The $r^*$ search algorithm for an optimal LSBT can be made to run in O($n \times \log^2 n$) time on a set of upload capacities $c$.*

**Example**. Figure 3.5 shows the illustration of searching results by applying Algorithm 3.2 to the example in Figure 3.2. The numerals in these gray areas in Figure 3.5

**Algorithm 3.2** The $r^*$ search algorithm for the optimal LSBT

**Input:** a set of upload capacities $c$ and *CandidateSet*
**Output:** $r^*$ **and** $D^*$

1: **BEGIN**
2: $CandidateSet \leftarrow Sort(CandidateSet)$
3: $D^* \leftarrow \infty$
4: $r^* \leftarrow empty$
5: **for** $h \leftarrow 1$ **to** $\lfloor \log_2 n \rfloor + 1$ **do**
6: $right \leftarrow 1$
7: $left \leftarrow Sizeof(CandidateSet)$
8: **while** $right \leq left$ **do**
9: $mid \leftarrow \lfloor (right + left)/2 \rfloor$
10: $r \leftarrow CandidateSet[mid]$
11: $t \leftarrow BuildLSBT(c, r)$
12: **if** $left - right = 1$ **and** $t.Height = h$ **then**
13: $d \leftarrow t.BroadcastingTime$
14: **if** $d < D^*$ **then**
15: $r^* \leftarrow r$
16: $D^* \leftarrow d$
17: **end if**
18: **else**
19: ***break***
20: **end if**
21: **if** $h \leq t.Height$ **then**
22: $right \leftarrow mid$
23: **else**
24: $left \leftarrow mid - 1$
25: **end if**
26: **end while**
27: **end for**
28: **return** $r^*$ **and** $D^*$
29: **END**

mean the heights of these LSBTs constructed by each different value of $r$. Recall that the given set of eleven nodes come with a set of upload capacities {3, 3, 2, 2, 2, 1, 1, 1, 1, 1, 1} in Figure 3.2. The numbered dashed lines specify the series of steps of the searching operation. According to **LEMMA** 3.3.1 and **LEMMA** 3.3.2, we can obtain the lower bound (*i.e.,* 0.3) and upper bound (*i.e.,* 1.8) in the example of Figure 3.2. As shown in Figure 3.5, the $CandidateSet$ consists of ten elements that returned by Algorithm 3.1. The points on horizontal axis presents the spectrum of the possible value of $r$ after the sorting operation, and the topmost curve specifies the maximum completion time $D$ procured by individual values of $r$. In the running example, we consider $h = 2$, *i.e.*, selecting the value of $r^*$ for

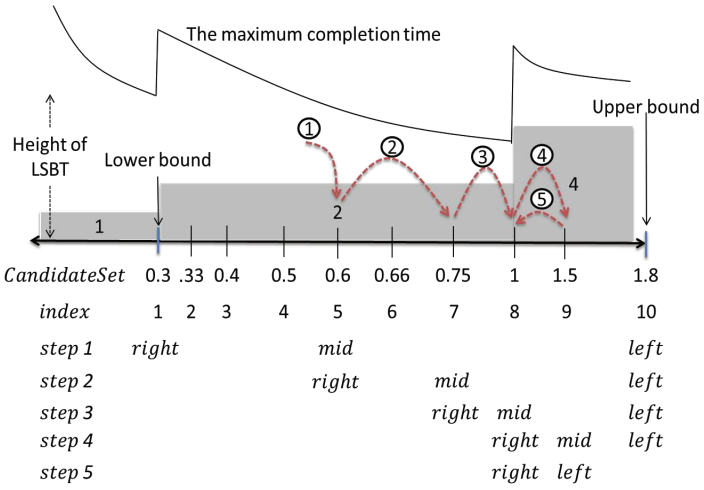



Figure 3.5: An illustration of the binary search algorithm for selecting the value of $r^*$ in the LSBT of height $h = 2$

the optimal LSBT with height 2. The bottom lines in Figure 3.5 illustrates the variances of $right$, $left$ and $mid$ in each step. The total number of step during the searching operation is five and the searching operation is terminated when $mid = right = 8$ and $left = 9$ (line 13 in Algorithm 3.2).

Finally, we show that the maximum number of rounds required to complete the broadcast of entire data chunks in an optimal LSBT is O$(m + logn)$ steps. The proof of **THEOREM** 3.3.6 requires to combine **THEOREM** 3.3.2 and **THEOREM** 3.3.4.

**THEOREM 3.3.6** *In an optimal LSBT, the minimum number of steps required to complete the broadcast of entire data chunks is O$(m+logn)$, where $m$ is the number of data chunks of equal size and $n$ presents the number of nodes in a network system.*

## 3.4 Numerical Evaluation

In this section, we analyze the performance of our LSBT through numerical evaluations. The algorithm developed in this chapter can be embodied in the control plane of big data service stacks to form node relationships that achieve the capacity. In the numerical results, we have implemented three approaches including FNF heuristic [38], DIM-Rank heuristic [46] and our LSBT. All of above heuristic algorithms are centralized, however in DIM-Rank, the cost of computing the broadcast schedule is non-trivial. Note that DIM-

Table 3.1: Node Uplink Capacity Distribution

| Uplink Distribution | | | | |
|---|---|---|---|---|
| Uplink Capacity (Kbps) | 128 | 384 | 1000 | 5000 |
| Fraction (%) | 20 | 40 | 25 | 15 |

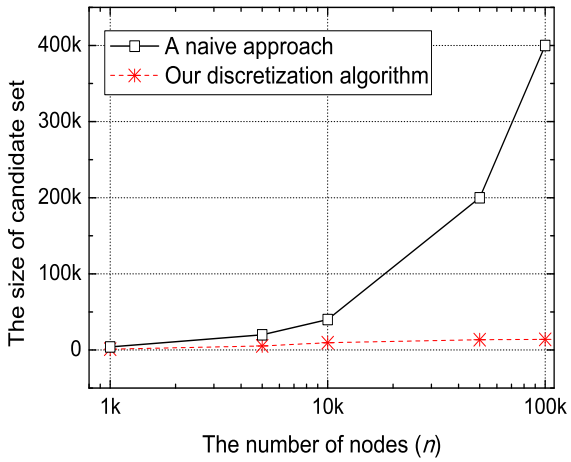
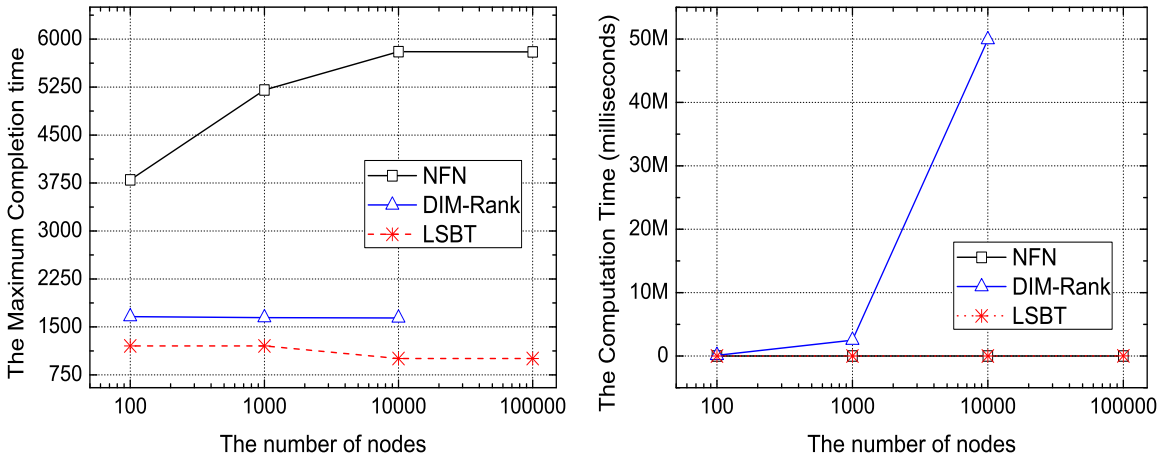



Figure 3.6: The size of candidateset versus the number of nodes $n$

Rank is the best algorithm in [46] by comparing other state-of-the-art algorithms. The node' uplink capacities distribution is set according to the actual Internet that is reported in [23] and their respective fractions in the node population are summarized in Table 3.1.

We first study the size of candidate set that derived from the Algorithm 3.1. Figure 3.6 shows the effect when the total number of nodes to be broadcasted is increased. Note that the $x$-axis is also a log-scale ($log_{10}n$). Algorithm 3.1 has a clear superior performance over the naive approach. Moreover, as the number of node is increased, the gap widens between Algorithm 3.1 and the naive approach making it very desirable. Intuitively, in the naive approach, the worst case of the size of $CandidateSet$ is the number of nodes multiplied by the size of $UnionSet$. Thus, the solution of Algorithm 3.1 may give a good heuristics for reducing the size of $CandidateSet$ in a large scale network.

We now show the maximum completion time of the three algorithms under various scenarios. We consider networks with $n$= 100, 1000, 10000 and 100000 nodes. The size of file is 100MB and the number of data chunks is 1000. Figure 3.7(a) shows the total time each algorithm taking to broadcast the file to all the nodes. Note that the $x$-axis is a log-scale of number of nodes and thus a straight line indicates good scalability, such as log-scale ($\log_1 0n$). Figure 3.7(b) shows the computation time of each algorithm to




(a) The Maximum Completion time

(b) The Computation Time

Figure 3.7: Performance comparison with increasing the number of nodes

schedule the broadcast job. By the simulation results, LSBT performs the best while FNF heuristics gives a poor performance, which is expected because FNF does not take the advantage of the pipeline manner. We notice that the computation time of DIM-Rank is significant, and it is because the time complexity of DIM-Rank is O$(m \times nlogn)^2$. In a $n = 10000$ network, the computation time of DIM-Rank requires almost 15 hours (on a sever with Intel Xeon 2.33GHz and 8GB RAM). Thus we do not plot the result of $n$ =100000 network.

Figure 3.8 plots the effect when the number of data chunks ($m$) is increased. The size of file is 100MB and the number of nodes is 100. We can see the result of FNF do not depend on the value of $m$. Note that the $x$-axis is also a log-scale ($log_2 m$). The maximum completion time of LSBT is significantly lower (at least about 60%) than the one performed by the two other algorithms. Another interesting remark is that DIM-Rank performs worse than FNF when $m = 2$. It is because in the concept of DIM-Rank algorithm, it prefers to let every node obtain a data chunk first. Thus the low-capacity nodes may slow down the maximum completion time.

## 3.5 Related work

The data broadcasting problem established by Edmonds [54] since the 1970s and has been studied in many articles. The broadcast problem is the core of every data distribu-

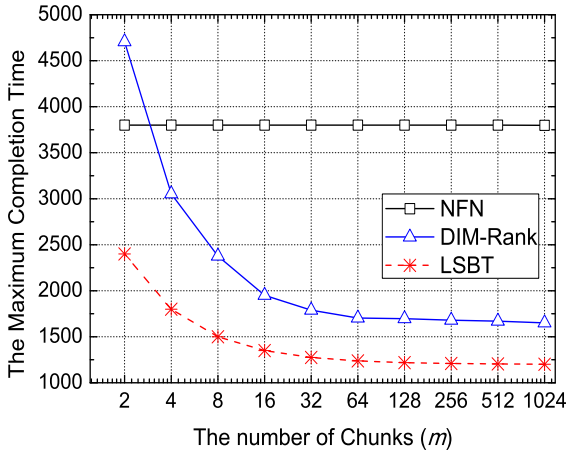
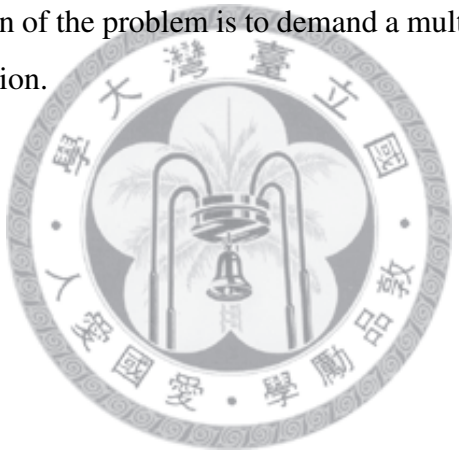



Figure 3.8: The number of chunks versus the maximum completion time

tion system, especially in peer-to-peer (P2P) overlay fields, it is of great interest to current efficient P2P data distribution systems, based on a tree or mesh design [4,5,47]. While there is much work on system design and measurement studies of P2P data distribution systems [48], few papers work on theoretical analysis and fundamental limitations of P2P data distribution systems. Ezovski *et al.* [55] proposed an optimal network topology and the associated scheduling policy to achieve the min-min times, by assuming that the file is broken into infinitesimally small chunks such that there is almost no forwarding delay. The authors claimed that the proposed scheme which achieves min-min times can also achieve the minimum average finish time. However, Chang *et al.* [56] disproved the claim in [55]. In [40], the authors propose several distributed algorithms to optimize the throughput of a broadcasting operation. However, they do not consider degree constraints in each node. In [41], Beaumont *et al.* considered the maximizing throughput problem of broadcasting a large message in heterogenous networks. They introduced the bounded degree multi-port model to model the capabilities of the nodes and proved that the data broadcasting problem of maximizing the overall throughput is NP-Complete. Liu *et al.* [57] studied the maximum streaming rate problem of peer-assisted streaming systems. They use a multi-tree formulation and consider per-tree degree bounds. However, they assume that the degrees of all nodes are equal, except for the source node which has unbounded degree. The same authors consider global per-node degree bounds in the article [58].

## 3.6 Conclusion

In this chapter, we studied the classical data broadcasting problem from an algorithmic point of view. We formalized the problem into a LockStep tree (LSBT) model in which we consider at the same time the design of such a single overlay tree (with a fixed uplink rate) and the maximum completion time of this model. To the best of our knowledge, this work is the first study to investigate the relation between a single overlay tree with a fixed uplink rate and the maximum completion time both in heterogeneous networks. In additional, We envisioned that our LSBT could be well-suited for a host of applications. We also proposed a novel polynomial-time algorithm to select the optimal uplink rate $r^*$ for building an optimal LSBT. The time complexity of our algorithm is O$(n \log^2 n)$. Interesting future work involves obtaining good heuristics to the data broadcasting problem. A more challenging version of the problem is to demand a multiple LSBTs, we leave it as an interesting future direction.



# CHAPTER 4

# Mobile Presence Dissemination: A Server Overlay Approach

"*Make things as simple as possible, but not simpler.*"

–Albert Einstein 1879-1955

BECAUSE of the ubiquity of the Internet, mobile devices and cloud computing environments can provide presence-enabled applications, *i.e.*, social network applications/services, worldwide. Facebook [11], Twitter [59], Google Latitude [60], buddycloud [61] and Mobile Instant Messaging (MIM) [62], are examples of presence-enabled applications that have grown rapidly in the last decade. Social network services are changing the ways in which participants engage with their friends on the Internet. They exploit the information about the status of participants including their appearances and activities to interact with their friends. Moreover, because of the wide availability of mobile devices (e.g., Smartphones) that utilize wireless mobile network technologies, social network services enable participants to share live experiences instantly across great distances. For example, Facebook receives more than 25 billion shared items every month and Twitter receives more than 55 million tweets each day. In the future, mobile devices will become more powerful, sensing and media capture devices. Hence, we believe it is inevitable that social network services will be the next generation of mobile Internet applications.

A mobile presence service is an essential component of social network services in cloud computing environments. The key function of a mobile presence service is to maintain an up-to-date list of presence information of all mobile users. The presence information includes details about a mobile user's location, availability, activity, device capability, and preferences. The service must also bind the user's ID to his/her current presence information, as well as retrieve and subscribe to changes in the presence information of

the user's friends. In social network services, each mobile user has a friend list, typically called a buddy list, which contains the contact information of other users that he/she wants to communicate with. The mobile user's status is broadcast automatically to each person on the buddy list whenever he/she transits from one status to the other. For example, when a mobile user logs into a social network application, such as an IM system, through his/her mobile device, the mobile presence service searches for and notifies everyone on the user's buddy list. To maximize a mobile presence service's search speed and minimize the notification time, most presence services use server cluster technology [63]. Currently, more than 500 million people use social network services on the Internet [11]. Given the growth of social network applications and mobile network capacity, it is expected that the number of mobile presence service users will increase substantially in the near future. Thus, a scalable mobile presence service is deemed essential for future Internet applications.

In the last decade, many Internet services have been deployed in distributed paradigms as well as cloud computing applications. For example, the services developed by Google and Facebook are spread among as many distributed servers as possible to support the huge number of users worldwide. Thus, we explore the relationship between distributed presence servers and server network topologies on the Internet, and propose an efficient and scalable server-to-server overlay architecture called PresenceCloud to improve the efficiency of mobile presence services for large-scale social network services.

First, we examine the server architectures of existing presence services, and introduce the buddy-list search problem in distributed presence architectures in large-scale geographically data centers. The *buddy-list search problem* is a scalability problem that occurs when a distributed presence service is overloaded with buddy search messages. Then, we discuss the design of PresenceCloud, a scalable server-to-server architecture that can be used as a building block for mobile presence services. The rationale behind the design of PresenceCloud is to distribute the information of millions of users among thousands of presence servers on the Internet. To avoid single point of failure, no single presence server is supposed to maintain service-wide global information about all users. PresenceCloud organizes presence servers into a quorum-based server-to-server architecture to facilitate efficient buddy list searching. It also leverages the server overlay and a

directed buddy search algorithm to achieve small constant search latency; and employs an active caching strategy that substantially reduces the number of messages generated by each search for a list of buddies. We analyze the performance complexity of PresenceCloud and two other architectures, a Mesh-based scheme and a Distributed Hash Table (DHT)-based scheme. Through simulations, we also compare the performance of the three approaches in terms of the number of messages generated and the search satisfaction which we use to denote the search response time and the buddy notification time. The results demonstrate that PresenceCloud achieves major performance gains in terms of reducing the number of messages without sacrificing search satisfaction. Thus, PresenceCloud can support a large-scale social network service distributed among thousands of servers on the Internet.

The following summarizes our contributions: First, PresenceCloud is among the pioneering architecture for mobile presence services. To the best of our knowledge, this is the first work that explicitly designs a presence server architecture that significantly outperforms those based distributed hash tables. PresenceCloud can also be utilized by Internet social network applications and services that need to replicate or search for mutable and dynamic data among distributed presence servers. The second contribution is that we analyze these scalability problems of distributed presence server architectures, and define a new problem called the buddy-list search problem. Through our mathematical formulation, the scalability problem in the distributed server architectures of mobile presence services is analyzed. Finally, we analyze the performance complexity of PresenceCloud and different designs of distributed architectures, and evaluate them empirically to demonstrate the advantages of PresenceCloud.

The remainder of this chapter is organized as follows. The next section contains a review of related works. In Section 4.1, we consider the buddy-list search problem in a distributed presence server architecture; and in Section 4.2, we describe the design of PresenceCloud in detail. The complexity analyses of PresenceCloud, the mesh-based scheme and the DHT-based scheme are presented in Section 4.3; and the performance results of the three approaches are detailed in Section 4.4. In Sections 4.5, we discuss performance issues related to PresenceCloud. Section 4.7 contains some concluding remarks.

## 4.1 The Problem Statement

In this section, we describe the system model, and the *buddy-list search problem*. Formally, we assume the geographically distributed presence servers to form a server-to-server overlay network, $G = (V, E)$, where $V$ is the set of the Presence Server (PS) nodes, and $E$ is a collection of ordered pairs of $V$. Each PS node $n_i \in V$ represents a Presence Server and an element of $E$ is a pair $(n_i, n_j) \in E$ with $n_i, n_j \in V$. Because the pair is ordered, $(n_j, n_i) \in E$ is not equivalent to $(n_i, n_j) \in E$. So, the edge $(n_i, n_j)$ is called an outgoing edge of $n_i$, and an incoming edge of $n_j$. The server overlay enables its PS nodes to communicate with one another by forwarding messages through other PS nodes in the server overlay. Also, we denote a set of the mobile users in a presence service as $U = \{u_1, \ldots, u_i, \ldots, u_m\}$, where $1 \leq i \leq m$ and $m$ is the number of mobile users. A mobile user $u_i$ connects with one PS node for search other user's presence information, and to notify the other mobile users of his/her arrival. Moreover, we define a *buddy list* as following.

**DEFINITION 4.1.1** ***Buddy list***, *$B_i = \{b_1, b_2, \ldots, b_k\}$ of user $u_i \in U$, is defined as a subset of $U$, where $0 < k \leq |U|$. Furthermore, $B$ is a symmetric relation, i.e, $u_i \in B_j$ implies $u_j \in B_i$.*

For example, given a mobile user $u_p$ is in the buddy list of a mobile user $u_q$, the mobile user $u_q$ also appear in the buddy list of the mobile user $u_p$. Note that to simplify the analysis of the Buddy-List Search Problem, we assume that buddy relation is a symmetric. However, in the design of PresenceCloud, the relation of buddies can be unilateral because the search operation of PresenceCloud can retrieve the presence of a mobile user by given the ID of the mobile user.

**PROBLEM STATEMENT:** ***Buddy-List Search Problem***: *When a mobile user $u_i$ changes his/her presence status, the mobile presence service searches presence information of mobile users in buddy list $B_i$ of $u_i$ and notifies each of them of the presence of $u_i$ and also notifies $u_i$ of these online buddies. The Buddy-List Search Problem is then defined as designing a server architecture of mobile presence service such that the costs of searching and notification in communication and storage are minimized.*

### 4.1.1 Analysis of a Naive Architecture of Mobile Presence Service:

In the following, we will give an analysis of the expected rate of messages generated to search for buddies of newly arrived user in a naive architecture of mobile presence services. We assume that each mobile user can join and leave the presence service arbitrarily, and each PS node only knows those mobile users directly attached to it. We also assume the probability for a mobile user to attach to a PS node to be uniform. Let's denote $\lambda$ the average arriving rate of mobile users in a mobile presence service. In this chapter, we focus on architecture design of mobile presence services and leave the problem of designing the capacity of presence servers as a separate research issue. Thus, we assume each PS node to have infinite service capacity. Hence, $\mu = \frac{\lambda}{n}$ is the average rate of mobile users attaching to a PS node, where $n$ is denoted the number of PS nodes in a mobile presence service. Let $h$ denote the probability of having all users in the buddy list of $u_i$ to be attaching to the same PS node as $u_i$. It is the probability of having no need to send search messages when $u_i$ attaches to a PS node. Thus,

$$h = \prod_{|B_i|} \frac{1}{n} = n^{-|B_i|}.$$

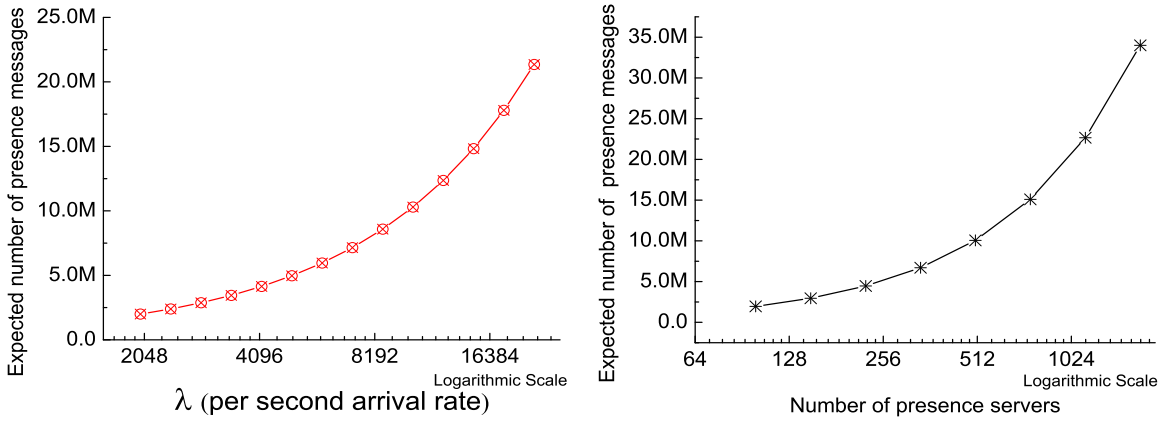
The expected number of search messages generated by this PS node per unit time is then

$$(n-1) \times (1-h) \times \mu.$$

For a reasonable size of set $B_i$ (e.g., $|B_i| \geq 3$) and $n \geq 100$, we consider the expected number $Q$ of messages generated by the $n$ PS nodes per unit time, then we have

$$\begin{aligned} Q &= n \times (n-1) \times (1-h) \times \mu \\ &= n \times (n-1) \times (1-h) \times \frac{\lambda}{n} \\ &\simeq (n-1) \times \lambda. \end{aligned}$$

Thus, as the number of PS nodes increase, both the total communication and the total CPU processing overhead of presence servers also increase. It also shows that $\lambda$ is another important parameter having impact on the system overhead. When $\lambda$ increases




(a) The average arrival rate of mobile users increases from 2,000 to 21,000 per second in a 1,000 PS nodes mobile presence system

(b) The number of PS nodes increases from 100 to 1,700 in a mobile presence service with 20,000 per second arrival rate of mobile users

Figure 4.1: The analysis of expected total transmissions in a mobile presence service

substantially, it has a major impact on the system overhead. However, a scalable presence system should be able to support more than 20,000 mobile user logins per second in burst cases as reported by [64].

In Fig. 4.1, we plot statistics for all expected queries transmitted in a mobile presence service to show the increase in number of buddy search messages as $\lambda$ increases. Note that the $x$ axis of both sub figures is in logarithmic scale. Fig. 4.1(a) shows that in a 1,000 PS nodes system, and the average arrival rate of mobile users increases from 2,000 to 21,000. From the results of analysis, the number of buddy searching messages increases with increasing average arrival rate of mobile users. Fig. 4.1(b) plots another scenario, the average arrival rate of user joining is 20,000 per second, and the number of PS nodes in a system increases from 100 to 1,700. It shows that the number of total buddy searching messages increases significantly with the number of PS nodes.

Collectively, we refer to the above phenomena as the *buddy-list search problem*, and any distributed presence server architectures could inevitably suffer from this scalability problem. The analysis shows that the buddy list searching operation of mobile presence services is costly and should be designed with caution.

## 4.2 Design of PresenceCloud

The past few years has seen a veritable frenzy of research activity in Internet-scale object searching field, with many designed protocols and proposed algorithms. Most of the previous algorithms are used to address the fixed object searching problem in distributed systems for different intentions. However, people are nomadic, the mobile presence information is more mutable and dynamic; a new design of mobile presence services is needed to address the buddy-list search problem, especially for the demand of mobile social network applications. PresenceCloud is used to construct and maintain a distributed server architecture and can be used to efficiently query the system for buddy list searches. PresenceCloud consists of three main components that are run across a set of presence servers. In the design of PresenceCloud, we refine the ideas of P2P systems and present a particular design for mobile presence services. The three key components of PresenceCloud are summarized below:

- **PresenceCloud server overlay** organizes presence servers based on the concept of *grid quorum system* [65]. So, the server overlay of PresenceCloud has a balanced load property and a two-hop diameter with $O(\sqrt{n})$ node degrees, where $n$ is the number of presence servers.

- **One-hop caching strategy** is used to reduce the number of transmitted messages and accelerate query speed. All presence servers maintain caches for the buddies offered by their immediate neighbors.

- **Directed buddy search** is based on the directed search strategy. PresenceCloud ensures an one-hop search, it yields a small constant search latency on average.

### 4.2.1 PresenceCloud Overview

The primary abstraction exported by our PresenceCloud is used to construct a scalable server architecture for mobile presence services, and can be used to efficiently search the desired buddy lists. We illustrated a simple overview of PresenceCloud in Fig. 4.2. In the mobile Internet, a mobile user can access the Internet and make a data connection to PresenceCloud via 3G or Wifi services. After the mobile user joins and authenticates

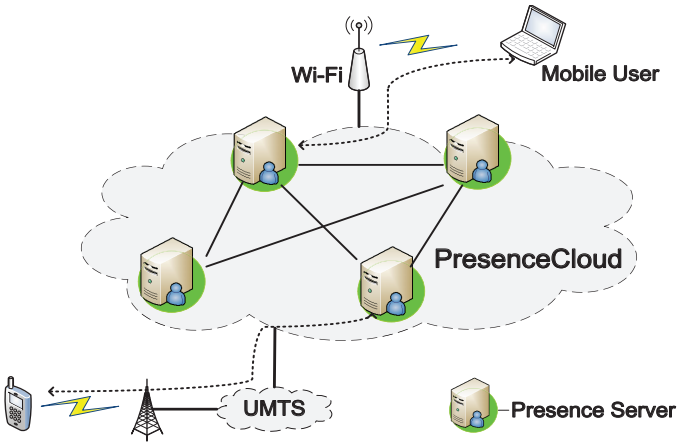



Figure 4.2: An overview of PresenceCloud

himself/herself to the mobile presence service, the mobile user is determinately directed to one of Presence Servers in the PresenceCloud by using the Secure Hash Algorithm, such as SHA-1 [66]. The mobile user opens a TCP connection to the Presence Server (PS node) for control message transmission, particularly for the presence information. After the control channel is established, the mobile user sends a request to the connected PS node for his/her buddy list searching. Our PresenceCloud shall do an efficient searching operation and return the presence information of the desired buddies to the mobile user. Now, we discuss the three components of PresenceCloud in detail below.

### 4.2.2 PresenceCloud Server Overlay

The PresenceCloud server overlay construction algorithm organizes the PS nodes into a server-to-server overlay, which provides a good low-diameter overlay property. The low-diameter property ensures that a PS node only needs two hops to reach any other PS nodes. The detailed description is as follows.

Our PresenceCloud is based on the concept of *grid quorum system* [65], where a PS node only maintains a set of PS nodes of size O($\sqrt{n}$), where $n$ is the number of PS nodes in mobile presence services. In a PresenceCloud system, each PS node has a set of PS nodes, called *PS list*, that constructed by using a grid quorum system, shown in Fig. 4.3 for $n$=9. The size of a grid quorum is $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$. When a PS node joins the server overlay of PresenceCloud, it gets an ID in the grid, locates its position in the grid and

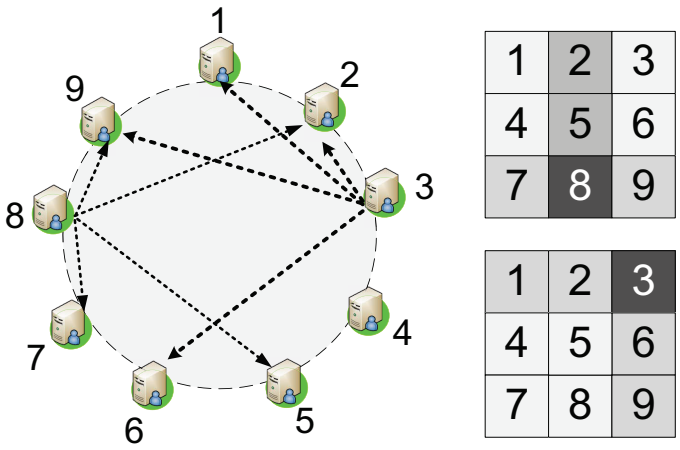



Figure 4.3: A perspective of PresenceCloud Server Overlay

obtains its PS list by contacting a root server[1]. On the $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ grid, a PS node with a grid ID can pick one column and one row of entries and these entries will become its PS list in a PresenceCloud server overlay. Fig. 4.3 illustrates an example of PresenceCloud, in which the grid quorum is set to $\lceil\sqrt{9}\rceil \times \lceil\sqrt{9}\rceil$. In the Fig. 4.3, the PS node 8 has a PS list {2,5,7,9} and the PS node 3 has a PS list {1,2,6,9}. Thus, the PS node 3 and 8 can construct their overly networks according to their PS lists respectively.

We now show that each PS node in a PresenceCloud system only maintains the PS list of size O($\sqrt{n}$), and the construction of PresenceCloud using the grid quorum results in each PS node can reach any PS node at most two hops.

**LEMMA 4.2.1** *The server overlay construction algorithm of PresenceCloud guarantees that each presence server only maintains a presence server list of size $O(\sqrt{n})$, where $n$ is the number of presence servers.*

**Proof**: We consider a grid quorum system of $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ and arrange elements of the global set $G = \{1, 2, \ldots, n\}$ as a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ array. For each grid point $i$ has a quorum set, $S_i$ that contains a full column plus a full row of elements in the array. Then it is clear that for any grid point $i$ in a grid quorum system, the size of $|S_i|$ is equal to $2\lceil\sqrt{n}\rceil - 1$.

[1]The root server may become a scalability problem in a very large scale PresenceCloud, *ex*, a million PS nodes. However, it also can be extended to work with distributed root servers, such as the KAD system in BitTorrent [67].

Therefore, for any PS node in a PresenceCloud, the size of a PS list maintained at a PS node is O($\sqrt{n}$). □

**LEMMA 4.2.2** *Each presence server in a PresenceCloud server overlay can reach any other presence servers in a two hops route.*

**Proof**: In a grid quorum system of $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$, for each grid point $i$ has a quorum set, $S_i$ that contains a full column plus a full row of elements in the array. It is clear that $S_i \cap S_j \neq null$ for any grid point $i$ and $j$, $1 \leq i, j \leq n$. Let $K$ be the some grid points in $S_i \cap S_j$. Thus, for any grid point, $j \notin S_i$, the grid point $i$ can reach $j$ by passing one of grid points, $l \in K$ set. Then it is clear that for any PS node $i$ in a PresenceCloud, can reach any other PS node in a two hops route. □

Here, we give an example for **LEMMA** 4.2.2. In Fig. 4.3, the PS list of PS node 1 is the set $\{2, 3, 4, 7\}$ and one of PS node 8 is the set $\{2, 5, 7, 9\}$. PS node 8 can reach PS node 1 by $K$ set $\{2, 7\}$, *i.g.*, a route $8 \rightarrow 2 \rightarrow 1$ or $8 \rightarrow 7 \rightarrow 1$. Note that $K$ set is the intersection set of two PS lists Consequently, PresenceCloud can hold the two-hop diameter property based on a grid quorum system.

**Stabilization of Incomplete Quorum Systems**

Our algorithm is fault tolerance design. At each PS node, a simple $Stabilization()$ process periodically contacts existing PS nodes to maintain the *PS list*. The $Stabilization()$ process is elaborately presented in the Algorithm 4.1. When a PS node joins, it obtains its PS list by contacting a root. However, if a PS node $n$ detects failed PS nodes in its PS list, it needs to establish new connections with existing PS nodes. In our algorithm, $n$ should pick a random PS node that is in the same column or row as the failed PS node.

In Algorithm 4.1, the function Find_CorrectPSNode() returns the correct PS list entry, *i.e.*, $findnode.id$ is equal to $pslist[i].id$. Moreover, it is an easy implemented function based on **LEMMA** 4.2.2. The function RandomNode($node$) is designed to pick a random PS node that is in the same column or row as the failed PS node, $node$. This function can retrieve substituted nodes by asking the existing PS node in PS list.

In our design, we define $pslist[i].id$ as the id of the $i$th logical neighbor of a PS node $p$, while $pslist[i].connection.id$ as the id of the physical PS node that handles presence

information for the $i$th logical neighbor of $p$. In other words, if the PS node with $ID = pslist[i].id$ does not exist on the overlay, then the Stabilization algorithm will assign the PS node with ID=$pslist[i].connection.id$ to take its place.

For example, in Fig. 4.3, we let PS node 8 be the node running the Stabilization algorithm. In general, in PS node 8, the values of $pslist[].id$ should be {2, 5, 7, 9} and the values of $pslist[].connection$ should be {node 2, node 5, node 7, node 9}. If the PS node 2 is failed then PS node 8 can choose node 1 or node 3 to take over PS node 2 after running the Stabilization algorithm. Clearly, several attractive features of our algorithm are that it is simple to implement, is naturally robust to failures, and is with a two-hop diameter property.

The heart beat messages overhead is another performance factor in distributed server overlays. Anyone distributed server overlay architecture requires heart beat message to maintain the connectivity of each server for recovering system from server failure. However, in order to reduce the maintenance overhead, PresenceCloud piggybacks the heart beat message in buddy search messages for saving transmission costs.

The following result quantifies the robustness of the PresenceCloud protocol.

**LEMMA 4.2.3** *if a presence server uses a PS list of size r in a PresenceCloud that is initially stable, and if every presence server fails with probability $\frac{1}{2}$, then the Stabilization algorithm sustains the PS list with high probability.*

**Proof**: Before any nodes fail, a node was aware of its $r$ immediate presence servers. The probability that all of these presence servers fail is $(\frac{1}{2})^r$, thus every presence server is aware of these presence servers in its PS list with high probability. As was implied by **LEMMA** 4.2.2, if the current node $n$ starts stabilization procedure and if the correct PS node $p$ exists, then node $n$ retrieves $p$ in one hop query. Therefore, the Stabilization algorithm can sustain the PS list by asking these immediate presence servers. □

Then, we describe a mobile user join procedure. When a mobile user joins a PresenceCloud, he/she uses a hash function, such as Secure Hash Algorithm (SHA-1) [66], to hash his/her identifier in the mobile presence service to get a grid ID in the grid quorum. PresenceCloud assigns mobile users to PS nodes with a hash function, e.g., SHA-1 algorithm such that, with a high probability, the hash function balances loads uniformly [68]

**Algorithm 4.1** PresenceCloud Stabilization algorithm

```
1: /* periodically verify PS node n's pslist */
2: Definition:
3: pslist: set of the current PS list of this PS node, n
4: pslist[].connection: the current PS node in pslist
5: pslist[].id: identifier of the correct connection in pslist
6: node.id: identifier of PS node node
7: Algorithm:
8: r ← Sizeof(pslist)
9: for i = 1 to r do
10:    node ← pslist[i].connection
11:    if node.id ≠ pslist[i].id then
12:       /* ask node to refresh n's PS list entries */
13:       findnode ← Find_CorrectPSNode(node)
14:       if findnode = nil then
15:          pslist[i].connection ← RandomNode(node)
16:       else
17:          pslist[i].connection ← findnode
18:       end if
19:    else
20:       /* send a heartbeat message */
21:       bfailed ← SendHeartbeatmsg(node)
22:       if bfailed = true then
23:          pslist[i].connection ← RandomNode(node)
24:       end if
25:    end if
26: end for
```

and thus all PS nodes receive roughly the same number of mobile users.

So that the mobile user can decide that he/she has to associate with the specific PS node, $q$ with the hashed grid ID. Note that PresenceCloud uses overlay network architecture. In other words, although the mobile user knows ID of the PS node $q$, it does not have IP address of $q$. In order to obtain IP address of $q$, the mobile user first hashes his/her identifier into the grid ID of the specific PS node $q$.

Then the mobile user contacts randomly with a of PS node, $p$ redirect he/she to the specific PS node, $q$. Note that PresenceCloud does not require a centralized server to response the IP address lookup of PS nodes for every user joining. However, if the specific PS node $q$ does not exist, the contacted PS node $p$ can redirect the mobile user to those PS nodes that are in the same row as the PS node $q$. For example, in Fig. 4.3, a mobile user obtains a grid ID 6 by hashing its identifier in the mobile presence service,

then he/she randomly contacts with PS node 2. And PS node 2 should redirect the mobile user to PS node 6. If PS node 6 is not existing, then PS node 2 should redirect the mobile user to PS node 4 or 5.

### 4.2.3 One-hop Caching

To improve the efficiency of the search operation, PresenceCloud requires a caching strategy to replicate presence information of users. In order to adapt to changes in the presence of users, the caching strategy should be asynchronous and not require expensive mechanisms for distributed agreement. In PresenceCloud, each PS node maintains a *user list* of presence information of the attached users, and it is responsible for caching the *user list* of each node in its PS list, in other words, PS nodes only replicate the *user list* at most one hop away from itself. The cache is updated when neighbors establish connections to it, and periodically updated with its neighbors. Therefore, when a PS node receives a query, it can respond not only with matches from its own *user list*, but also provide matches from its caches that are the user lists offered by all of its neighbors.

Our caching strategy does not require expensive overhead for presence consistency among PS nodes. When a mobile user changes its presence information, either because it leaves PresenceCloud, or due to failure, the responded PS node can disseminate its new presence to other neighboring PS nodes for getting updated quickly. Consequently, this one-hop caching strategy ensures that the user's presence information could remain mostly up-to-date and consistent throughout the session time of the user.

More specifically, it should be easy to see that, each PS node maintains roughly $2(\lceil\sqrt{n}\rceil - 1) \times u$ replicas of presence information, due to each PS node replicates its *user list* at most one hop away from itself. Here, $u$ is denoted the average number of mobile users in a PS node. Therefor, we have the following lemma.

**LEMMA 4.2.4** *Each presence server in PresenceCloud only maintains the one-hop replicas of presence information of size $O(u \times \sqrt{n})$, where $u$ is denoted the average number of mobile users in a presence server and $n$ is the number of presence servers.*

By maintaining O$(u \times \sqrt{n})$ replicas of presence information at each PS node and the simple two-hop overlay design, PresenceCloud has sufficient redundancy to provide

a good level of buddy searching service. Furthermore, this caching mechanism can significantly reduce the communication cost during searching operation. In the next section, an analysis studies will reveal that the one-hop caching mechanism brings PresenceCloud great improvement in buddy searching cost.

### 4.2.4 Directed Buddy Search

We contend that minimizing searching response time is important to mobile presence services. Thus, the buddy list searching algorithm of PresenceCloud coupled with the two-hop overlay and one-hop caching strategy ensures that PresenceCloud can typically provide swift responses for a large number of mobile users. First, by organizing PS nodes in a server-to-server overlay network, we can therefore use one-hop search exactly for queries and thus reduce the network traffic without significant impact on the search results. Second, by capitalizing the one-hop caching that maintains the user lists of its neighbors, we improve response time by increasing the chances of finding buddies. Clearly, this mechanism both reduces the network traffic and response time. Based on the mechanism, the population of mobile users can be retrieved by a broadcasting operation in any PS node in the mobile presence service. Moreover, the broadcasting message can be piggybacked in a buddy search message for saving the cost.

As previously mentioned, PresenceCloud does not require a complex searching algorithm, the directed searching technique can improve search efficiency. Thus, we have the following lemma.

**LEMMA 4.2.5** *For each buddy list searching operation, the directed buddy search of PresenceCloud retrieves the presence information $\Phi$ of the queried buddy list at most one-hop.*

**Proof**: This is a direct consequence of **LEMMA** 4.2.2 and **LEMMA** 4.2.3. □

Before presenting the directed buddy search algorithm, lets revisit some terminologies which will be used in the algorithm.

$B = \{b_1, b_2, \ldots, b_k\}$: set of identifiers of user's buddies

$B^{(i)}$: Buddy List Search Message be sent to PS node $i$

$b^{(i)}$: set of buddies that shared the same grid ID $i$

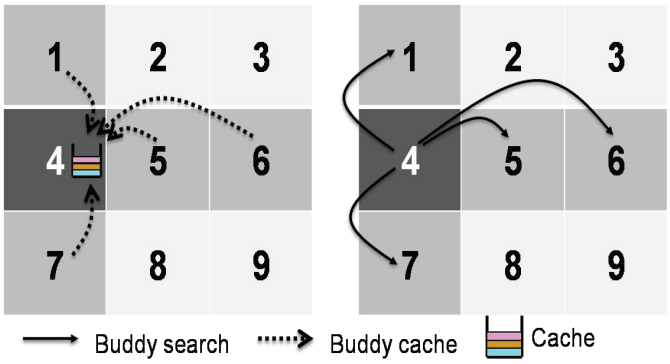



Figure 4.4: An example of buddy list searching operations in PresenceCloud

$S_j$: set of $pslist[].id$ of PS node $j$

**Directed Buddy Search Algorithm:**

1. A mobile user logins PresenceCloud and decides the associated PS node, $q$.
2. The user sends a Buddy List Search Message, $B$ to the PS node $q$.
3. When the PS node $q$ receives a $B$, then retrieves each $b_i$ from $B$ and searches its user list and one-hop cache to respond to the coming query. And removes the responded buddies from $B$.
4. If $B = nil$, the buddy list search operation is done.
5. Otherwise, if $B \neq nil$, the PS node $q$ should hash each remaining identifiers in $B$ to obtain a grid ID, respectively.
6. Then the PS node $q$ aggregates these $b^{(g)}$ to become a new $B^{(j)}$, for each $g \in S_j$. Here PS node $j$ is the intersection node of $S_q \cap S_g$. And sends the new $B^{(j)}$ to PS node $j$.

Following, we describe an example of directed buddy search in PresenceCloud. When a PS node 4 receives a Buddy List Search Message, $B = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, from a mobile user, PS node 4 first searches its local *user list* and the buddy cache, and then it responds these searched buddies to the mobile user and removes these searched buddies from $B$. In Fig. 4.4, these removed buddies include the user lists of PS node {1,4,5,6,7}. Then PS node 4 can aggregates $b^{(3)}$ and $b^{(9)}$ to become a new $B^{(6)}$ and sends the new $B^{(6)}$ to PS node 6. Note that the $pslist[].id$ of PS node 6 is {3,4,5,9}. Here, PS node 4 also aggregates $b^{(2)}$ and $b^{(8)}$ to become a new $B^{(5)}$ and sends the new $B^{(5)}$ to PS node 5. However, due to the one-hop caching strategy, PS node 6 has a buddy cache that

contains these user lists of PS node {3,9}, PS node 6 can expeditiously respond the buddy search message $B^{(6)}$. Consequently, the directed searching combined with both previous two mechanisms, including PresenceCloud server overlay and one-hop caching strategy, can reduce the number of searching messages sent.

## 4.3 Cost Analysis

In this section, we provide a cost analysis of the communication cost of PresenceCloud in terms of the number of messages required to search the buddy information of a mobile user. Note that how to reduce the number of inter-server communication messages is the most important metric in mobile presence service issues. The *buddy-list search problem* can be solved by a brute-force search algorithm, which simply searches all the PS nodes in the mobile presence service. In a simple mesh-based design, the algorithm replicates all the presence information at each PS node; hence its search cost, denote by $Q_{Mesh}$, is only one message. On the other hand, the system needs $n-1$ messages to replicate a user's presence information to all PS nodes, where $n$ is the number of PS nodes. The communication cost of searching buddies and replicating presence information can be formulated as $M_{cost} = Q_{Mesh} + R_{Mesh}$, where $R_{Mesh}$ is the communication cost of replicating presence information to all PS nodes. Accordingly, we have $M_{cost}$ =O$(n)$.

In the analysis of PresenceCloud, we assume that the mobile users are distributed equally among all the PS nodes, which is the worst case of the performance of PresenceCloud. Here, the search cost of PresenceCloud is denoted as $Q_p$, which is $2 \times (\lceil \sqrt{n} \rceil - 1)$ messages for both searching buddy lists and replicating presence information. Because search message and replica message can be combined into one single message, the communication cost of replicating, $R_p = 0$. It is straightforward to know that the communication cost of searching buddies and replicating presence information in PresenceCloud is $P_{cost} = Q_p = 2 \times (\lceil \sqrt{n} \rceil - 1)$. However, in PresenceCloud, a PS node not only searches a buddy list and replicates presence information, but also notifies users in the buddy list about the new presence event. Let $b$ be the maximum number of buddies of a mobile user. Thus, the worst case is when none of the buddies are registered with the PS nodes reached by the search messages and each user on the buddy list is located on

different PS nodes. Since PresenceCloud must reply every online user on the buddy list individually, it is clear that extra $b$ messages must be transmitted. In the worst case, it needs other $2 \times (\lceil \sqrt{n} \rceil - 1)$ messages (when $b >> 2(\lceil \sqrt{n} \rceil - 1)$). When all mobile users are distributed equally among the PS nodes, which is considered to be the worst case, the $P_{cost} = 4 \times (\lceil \sqrt{n} \rceil - 1)$. Consequently, we have **LEMMA** 4.3.1 and **THEOREM** 4.3.1.

**LEMMA 4.3.1** *For each buddy searching operation in PresenceCloud, the maximum communication cost of the buddy list search problem is $O(\sqrt{n})$, where $n$ is the number of presence servers.*

**THEOREM 4.3.1** *In a stable PresenceCloud of n nodes, the average notification messages caused by a buddy searching operation is equal to $b - \frac{2b}{\lceil \sqrt{n} \rceil}$, where $b$ is the maximum number of buddies of a user.*

**Proof**: In a stable PresenceCloud of $n$ nodes, the originator PS node forwards the buddy search event to its *PS list* with one hop. Then these PS nodes in its *PS list* may relay the extra notify event with another hop. Thus, the extra notification messages are only caused in the last hop. The size of *PS list* is $2\lceil \sqrt{n} \rceil - 1$. Thus, the average notification messages caused by a buddy searching operation in PresenceCloud is approximately equal to $b - \frac{b}{n} - \frac{b \times (2\lceil \sqrt{n} \rceil - 1)}{n} = \frac{(n-1-(2\lceil \sqrt{n} \rceil - 1)) \times b}{n} = b - \frac{2b}{\lceil \sqrt{n} \rceil}$. □

Next, we discuss the search cost of the DHT-based presence architecture. We make the following assumptions to simplify the analysis: 1) the presence information of a mobile user is only stored in one PS node (i.e. no replication). Note that in some DHT systems, replicating data increases both the node workload and the maintenance complexity. 2) all mobile users are uniformly distributed in all PS nodes. Although our analysis is based on the Chord [69], it can be extended to other DHTs. Let $n$ be the total number of PS nodes in a Chord. Chord nodes maintains $logn$ neighbors, *i.e.*, finger table, to provide a O($logn$) lookup operations. However, the lookup operation in DHT systems is based on exact-matching, each buddy must be searched one by one, the total search complexity of DHT is equal to $D_{cost}$ = O($b \times \log n$). Note that some replica algorithms [70] are proposed for DHT systems, but these algorithms also increase the complexity of DHT.

**Example**. We Consider the following simple example to illustrate the efficiency of PresenceCloud. Assume that there are 1024 ($n$=1024) PS nodes in PresenceCloud and the

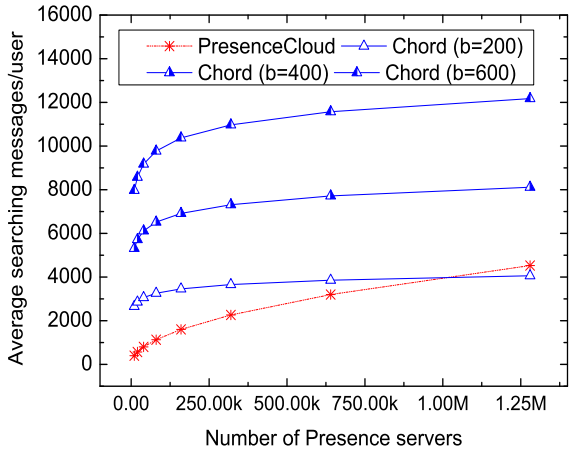



Figure 4.5: Average searching messages vs. very large number of PS nodes

maximum number of buddies is 100 ($b$=100). When a mobile user queries, the expected value of the number of messages which a PS node involves is less than (4×($\lceil\sqrt{1024}\rceil$ - 1)) = 124. It means that our PresenceCloud saves (124/1023) = 88% communication cost over the mesh-based approach. Then, we also have an example of DHT-based presence system, we assume that there are 1024 PS nodes in the DHT-based presence system and the maximum number of messages which a PS node involves is ($100 \times \log_2 1024$) = 1000 per user querying operation in the worst case (each buddy of the user is attached to different PS nodes).

Fig. 4.5 plots the average number of searching messages per searching operation in various very larger number of PS nodes, where $b$ is the number of buddy sizes. As expected, the average message transmissions of Chord-based design increases with the buddy size. The reason is that based on the traditional DHT protocol, each buddy should be treated as a key by hash functions for routing operation in most of DHTs. And it also needs at most $b$ messages for reporting the searching results to the source PS node. We can see that in the case ($b$=200), when the number of PS nodes grows, PresenceCloud performs better performance under one million PS nodes than Chord-based. However, PersenceCloud outperforms than Chord-based approach when the buddy size grows larger. Note that mesh architecture generates much more searching messages per user than the other architectures. It is thus difficult to put the comparisons in a single figure.

We summarize the comparison of different schemes in Table 4.1. The columns show

Table 4.1: Presence Architecture Comparison

| | Mesh | PresenceCloud | DHT-based |
|---|---|---|---|
| Search | O($n$) | O($\sqrt{n}$) | O($b \times \log n$) |
| Replicas | O($\|U\|$) | O($\sqrt{n} \times u$) | O($u$) |
| Latency | one hop | two hops | $\log n$ hops |
| Message Size | $b$ | $b/n$ | $b/n$ |
| Message Reply Hops | O(1) | O(1) | O($\log n$) |
| Maintenance Overhead | O($n$) | O($\sqrt{n}$) | O($\log n$) |

the different schemes. The label "Search" means the maximum number of messages sent by a PS node when a mobile user joins; label "Replica" means the maximum number of buddy replicas in a PS node. label "Latency" means the buddy search latency, we quantify this metric by the diameter of the server overlay. This is reasonable because, in general, the search latency is dominated by the diameter of the overlay. label "Message Size" means the average number of required buddy in a message. label "Message Reply Hops" means that the maximum hop counts of a message relayed by PS nodes. label "Maintenance Overhead" means the number of PS nodes maintained by a PS node. In Table 4.1, none of the schemes is a clear winner. The mesh-based approach achieves good search latency at the expense of the other metrics. Our PresenceCloud yields a low communication cost in large-scale server architecture and small search latency. Note that $u$ is denoted the average number of mobile users attached to a PS node and $b$ is the number of buddy sizes. Meanwhile, the DHT-based method provides good features for low replica load, however it comes at a prices of increased searching latency.

## 4.4 Performance Evaluation

In this section, we present the details of the framework used for the experiments. Our implementation of the network simulator and the related architectures, including a Mesh-based, PresenceCloud and a Chord-based presence server architecture, was written in Java. The packet-level simulator allows us to perform tests up to 20,000 users and 2,048 PS nodes, after which simulation data no longer fit in RAM and makes our experiments difficult. In our experiments, the simulator first goes through a warming-up phase to reach

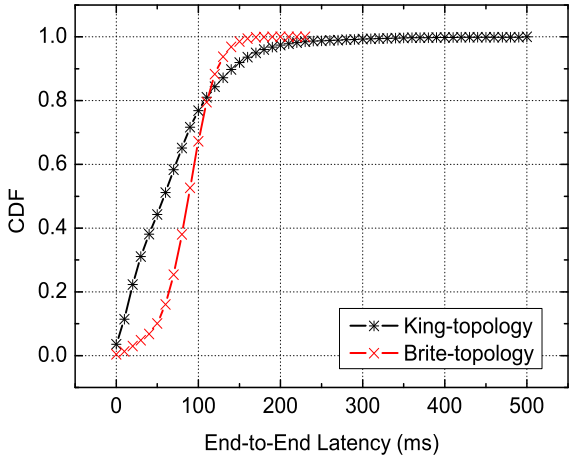



Figure 4.6: End-to-end latency distribution of King topology and Brite-topology

the network size (both PS nodes and users), and the simulator starts of the 1,800 seconds test after the measurement approach has stabilized (the stabilized time is based on the system size).

We apply two physical topologies to simulate Internet networks. 1) King-topology: This is a real Internet topology from the King data set. The King data set delay matrix is derived from Internet measurements using techniques that described by Gummadi *et al.* [71]. It consists of 2,048 DNS servers. The latencies are measured as RTTs between the DNS servers. 2) Brite-topology: This is an AS topology generated by the BRITE topology generator [72] using the Waxman model where alpha and beta are set to 0.15 and 0.2, respectively. In addition, HS (size of one side of the plane) is set to 1,000 and LS (size of one side of a high-level square) is set to 100. Totally, the Brite-topology consists of 1,000 nodes. In Fig. 4.6, we show the CDF of the King-topology and the CDF of the Brite-topology.

The simulated topology places every PS node at a position on the King-topology or the Brite-topology, chosen uniformly at random. Note that our simulations involve networks of less than 2,048 PS nodes, we use a pairwise latency matrix derived from measuring the inter-PS node latencies. And each mobile user also uniformly is attached to a random PS node, the propagation delay between mobile user and PS node is randomly assigned in the range [1,20] (ms). The King-topology is assumed as the default IP network topology. Every simulation result is the average 20 runs. The average delay of King-

topology is 77.4 milliseconds and 96.2 milliseconds in the Brite-topology. Therefore, the number of users is set to be 20,000, unless otherwise specified.

Experiments were preformed on a Intel 2.8GHz Pentium machine with 4G RAM. The rest of this section is organized as follows. In Section 4.4.1, we discuss the three important criteria using in the evaluation. Finally, we report the performance results of the three server architectures.

### 4.4.1 Performance Metrics

Within the context of the model, we measure the performance of server architectures using the following three metrics: *1) Total Searching Messages:* This represents the total number of messages transferred between the query initiator and the other PS nodes during the simulation time. This is meat and potatoes metric in our experiments, since it is widely regarded to be critical in a mobile presence service that we discussed both in the Section 4.1 and the Section 4.3. *2) Average Searching Messages per-arrived user:* The number of searching messages used per arrived user. Moreover, this metric is independent of user arrival pattern. *3) Average Searching latency:* This represents that average buddy searching time for a joining mobile user. This metric is a critical metric for measuring the search satisfaction of mobile presence services.

### 4.4.2 Simulation Results

We first evaluate and compare the three server architectures by considering the total buddy searching messages metric. We instantiated a server network of 256 PS nodes in our simulator, and ran a number of experiments to investigate the effect of scalability of PS nodes on involved searching messages. More precisely, we varied the user arrival rate from 100 per second to 8,000 per second to explore the relation between user arrival rate and the total searching messages. In this test, the number of buddies is set to 100.

Fig. 4.7 depicts the total number of searching message transmissions during simulation time (1,800 seconds) under various rates of user arrival patterns (100 to 8,000 per second). As the analytical results we discussed in Section 4.4, we shows that for a given number of PS nodes, the total number of searching messages is dominated by the user

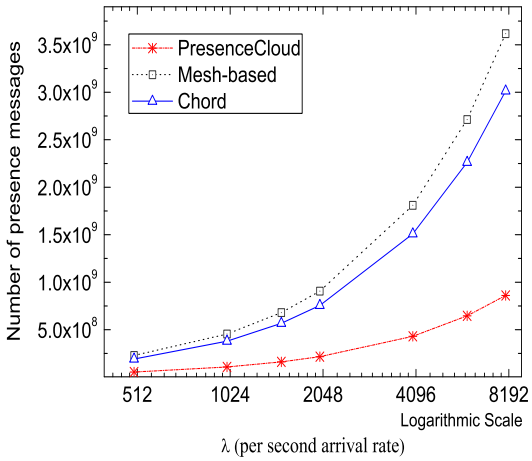



Figure 4.7: The total message transmissions during simulation time (1,800s)

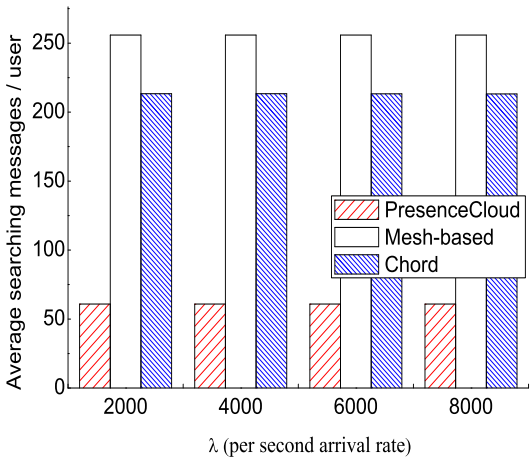



Figure 4.8: The average message transmissions per searching operation

arrival rate ($\lambda$) significantly. In Fig. 4.7, the total number of searching messages significantly increased as the user arrival rate increased. We could see that PresenceCloud outperforms all other designs. Mesh-base and Chord both require an enormous number of messages for searching buddy lists in higher user arrival rates. However, vast message transmissions may limit the scalability of the server architecture in mobile presence services.

Fig. 4.8 shows the average number of searching message transmissions during simulation time (1,800 seconds) under various rates of user arrival patterns (100 to 8,000 per second). As shown in Fig. 4.8, the average number of searching message transmissions is independent of user arrival pattern. Increasing the rate of user arrival pattern does not increase the average searching message transmissions. For each design, the number of message transmissions is bounded as shown as Section 4.3. Our PersenceCloud requires the least message transmissions. But mesh-based requires O($n$) searching complexity (note that the number of PS node is set to 256), the experimental results fit our analysis in the Section 4.3. Chord-based design performs second highest message transmissions per searching operation. However, if the server architecture is not designed well, the scalability problem of servers may limit itself to scale more than thousands size, hence a poor server architecture may not support a very large number of servers.

In order to study the scalability of server architecture designs to the number of severs, we ran experiments in which the user arrival rate is fixed to 2,000 per second and

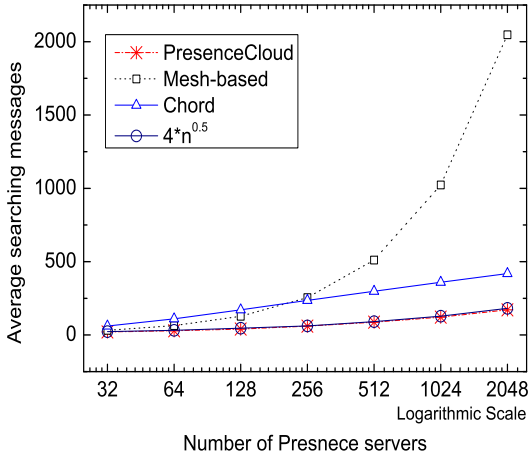



Figure 4.9: Average searching messages vs. number of PS nodes

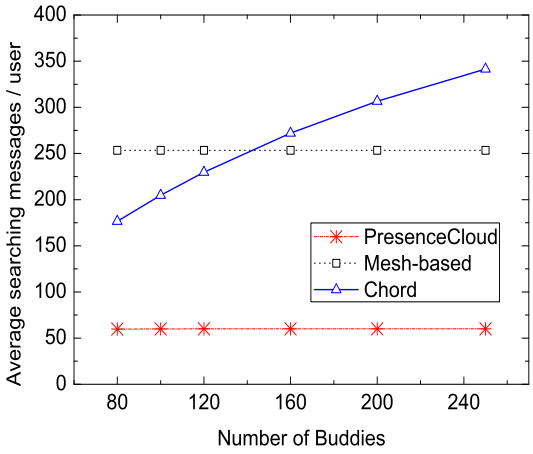



Figure 4.10: Average searching messages vs. number of buddy in a 256 PS nodes system

the number of buddies is set to 120. In these experiments, we increase the number of presence sever nodes from 32 to 2,048. Fig. 4.9 plots the average number of searching messages per searching operation in various number of PS nodes. As expected, the average message transmissions of PresenceCloud increases gradually with the number of servers. However, the average message transmissions of PresenceCloud is bounded by $4 \times \sqrt{n}$. Recall that we had shown the searching complexity of our PresenceCloud in the Section 4.3. This results suggest good scalability with the number of servers for server architecture design. The simulation results also verify our analysis. Moreover, the mesh-based performs the poorest performance than other sever architecture designs. It requires O($n$) searching complexity. Chord-based performs logarithmical performance in this metrics. Generally, the number of average message transmissions grows slowly with the network size in Chord-based and PresenceCloud designs.

In the following we studied the scalability of server architecture designs while varying the number of buddies per mobile user. We ran experiments in which the number of PS nodes is set to 256 and the user arrival rate is fixed to 2,000 per second. In these experiments, we increase the number of buddies per user from 80 to 250. Fig. 4.10 plots the average number of searching messages per searching operation in various number of buddy per mobile user. As expected, the average message transmissions of PresenceCloud and mesh-based are not impacted by the buddy size. However, the average message transmissions of Chord-based design increases with the buddy size. The reason is that based

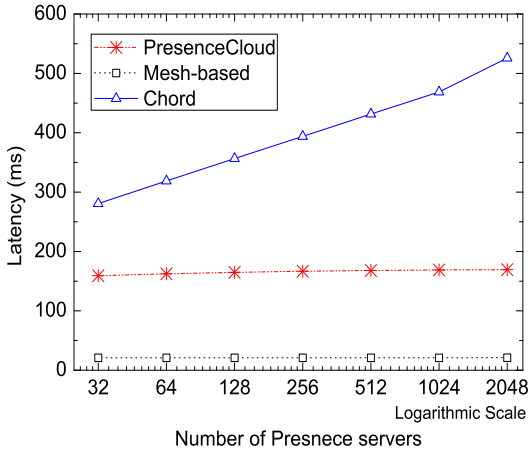



Figure 4.11: Average buddy searching latency vs. number of PS nodes

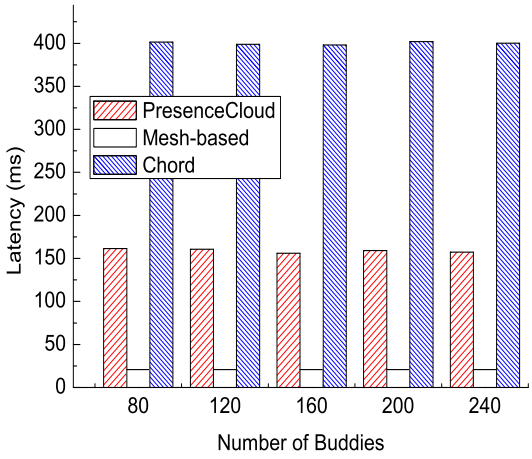
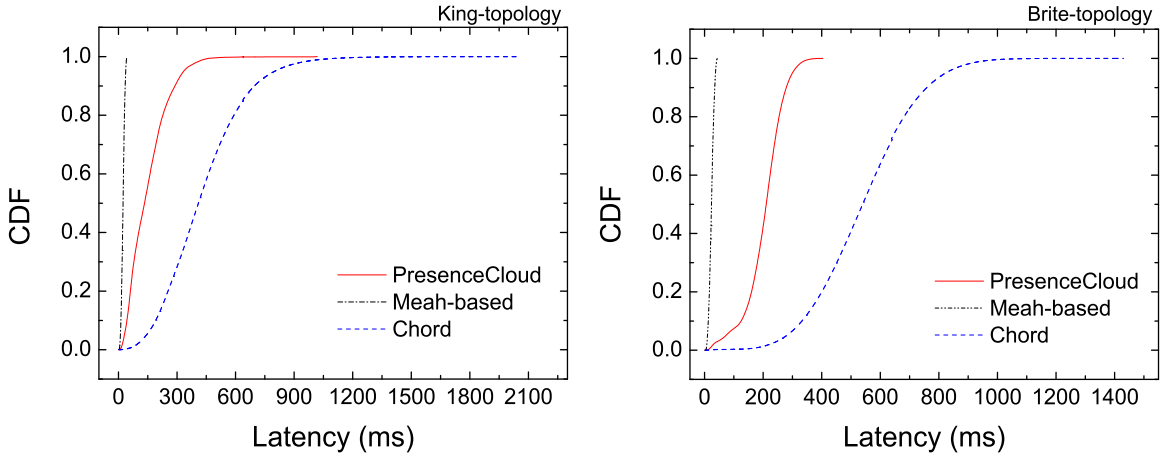



Figure 4.12: Average buddy searching latency vs. number of PS nodes

on the traditional DHT protocol, each buddy should be treated as a key by hash functions for routing operation in most of DHTs. And it also needs at most $b$ messages for reporting the searching results to the source PS node.

Next, we investigate the search satisfaction of server architecture designs. We use our simulator to study the buddy searching latency while varying the number of PS nodes. The simulation environment is set as Fig. 4.9. As shown as Fig. 4.11, for PresenceCloud, the buddy searching latency grows gently with the number of PS nodes. However, the buddy searching latency of mesh-based design is significantly better than PresenceCloud. The reason is that, by using the mesh-based design, every PS node can retrieve all desired buddy information in its current replica and return the presence information of buddy to user in one hop RTT, and the one hop RTT is quite small in our assumption. PresenceCloud, on the other hand, needs to retrieve all available replicas from its neighbors, which affects the buddy search time. Although the mesh-based design achieves a faster buddy search time and a higher replica hit ratio than PresenceCloud, it sacrifices the scalability of the server architecture in mobile presence services. Under the Chord-based design, a search operation may need to visit a logarithmic number of PS nodes to find the buddies of users. Thus, for latency-sensitive applications, DHT-based designs may be unsuitable for mobile presence services due to their high lookup costs [73].

We also studied the buddy searching latency of server architecture designs while varying the number of buddies. The simulation environment is set as Fig. 4.10. In these




(a) King-topology (b) Brite-topology

Figure 4.13: The CDF of buddy searching latency in King and Brite topology

experiments, we increase the number of buddies per user from 80 to 240. Fig. 4.12 depicts the buddy searching latency with the addition of buddy until 240. As shown in Fig. 4.12, in all designs, the buddy searching latency is not impacted by the number of buddies. The search latency is dominated by the diameter of the overlay, thus the buddy searching latency does not grows with the number of buddies. Clearly, it is a tradeoff, the experiment results show that mesh-based design performs best search satisfaction, but suffers heavily communication cost. However, our PresenceCloud reduces the significantly communication cost without sacrificing search satisfaction extremely.

Note that the buddy searching latency is a critical metric for measuring the search satisfaction of a mobile presence service. To the best of our knowledge, we are not aware any study about the buddy searching latency in mobile presence services. However, in our survey, we noticed that the average DNS lookup latency was 255.9 ms that reported by Ramasubramanian *et al.* [74]. The results is estimated in a large scale DNS in Planet Lab. This report could become a solid reference material for user satisfaction. Compared to the DNS lookup results in the article, the buddy searching latency of PresenceCloud is tolerable. With high probability, we could expect that our PresenceCloud appeases the user satisfaction basically.

Fig. 4.13 plots that the relative performance of various server architecture designs in the buddy searching latency for two different network topologies. We ran experiments in which the number of PS nodes is set to 512, the user arrival rate is fixed to 2,000 per sec-

ond, and the number of buddies is set to 200. Fig. 4.13(a) shows the latency distribution results for the king-topology, while Fig. 4.13(b) shows the results for the Brite-topology specified earlier. As shown in the two Figures 4.13(a) and 4.13(b), the mesh-based design takes much less time to search buddy list than all two other designs under both King and Brite topologies. With the King-topology, the curve for Chord-based design has a flatter tail. This is due to the fact that the distribution of physically link latency in king-topology is more skewed. Note that the mean delay in King-topology is 77.4 milliseconds and is 96.2 milliseconds in the Brite-topology. Thus the expected search latency for PresenceCloud is about 150 milliseconds in King-topology and is approximately 200 milliseconds in Brite-topology. However, the high delay results of Chord-based design are caused by searches whose hops follow high delay paths.

## 4.5 Discussions

A number of issues require further consideration. Our current PresenceCloud does not address the communication security problem, and the presence server authentication problem, we discuss the possible solutions as follows. The distributed presence service may make the mobile presence service more prone to communication security problems, such as malicious user attacks and the user privacy. Several approaches are possible for addressing the communication security issues. For example, the Skype protocol offers private key mechanisms for end-to-end encryption. In PresenceCloud, the TCP connection between a presence server and users, or a presence server could be established over SSL to prohibit user impersonation and man-in-the-middle attacks. This end-to-end encryption approach is also used in XMPP/SIMPLE protocol.

The presence server authentication problem is another security problem in distributed presence services. In centralized presence architectures, it is no presence server authentication problem, since users only connect to an authenticated presence server. In PresenceCloud, however, requires a system that assumes no trust between presence servers, it means that a malicious presence server is possible in PresenceCloud. To address this authentication problem, a simple approach is to apply a centralized authentication server. Every presence server needs to register an authentication server; PresenceCloud could

certificate the presence server every time when the presence server joins to PresenceCloud. An alternative solution is PGP web of trust model [75], which is a decentralized approach. In this model, a presence server wishing to join the system would create a certifying authority and ask any existing presence server to validate the new presence server's certificate. However, such a certificate is only valid to another presence server if the relying party recognizes the verifier as a trusted introducer in the system. These two mechanisms both can address the directory authentication problem principally.

In additional, the user satisfaction of mobile presence service is another search issue. Several studies have investigated the issues of user satisfaction in several domains, including VOIP [76], WWW search engine [77]. To the best of our knowledge, there is no study of exploring the user satisfaction issues, such as search response time, search precise, etc, about mobile presence services. Given the growth of social network applications and mobile device computing capacity, it is an interesting research direction to explore the user satisfaction both on mobile presence services or mobile devices.

## 4.6 Related Work

In this section, we describe previous researches on presence services, and survey the presence service of existing systems. Well known commercial IM systems leverage some form of centralized clusters to provide presence services [63]. Jennings III *et al.* [63] presented a taxonomy of different features and functions supported by the three most popular IM systems, AIM, Microsoft MSN and Yahoo! Messenger. The authors also provided an overview of the system architectures and observed that the systems use client-server-based architectures. Skype, a popular voice over IP application, utilizes the Global Index (GI) technology [78] to provide a presence service for users. GI is a multi-tiered network architecture where each node maintains full knowledge of all available users. Since Skype is not an open protocol, it is difficult to determine how GI technology is used exactly. Moreover, Xiao *et al.* [79] analyzed the traffic of MSN and AIM system. They found that the presence information is one of most messaging traffic in instant messaging systems. In [80], authors shown that the largest message traffic in existing presence services is buddy NOTIFY messages.

Several IETF charters [81–83] have addressed closely related topics and many RFC documents on instant messaging and presence services have been published, such as XMPP [84] and SIMPLE [85]. Jabber [86] is a well-known deployment of instant messaging technologies based on distributed architectures. It captures the distributed architecture of SMTP protocols. Since Jabber's architecture is distributed, the result is a flexible network of servers that can be scaled much higher than the monolithic, centralized presence services. Recently, there is an increase amount of interest in how to design a peer-to-peer SIP [87]. P2P-SIP [88] has been proposed to remove the centralized server, reduce maintenance costs, and prevent failures in server-based SIP deployment. To maintain presence information, P2PSIP clients are organized in a DHT system, rather than in a centralized server. However, the presence service architectures of Jabber and P2PSIP are distributed, the *buddy-list search problem* we defined later also could affect such distributed systems.

It is noted that few articles in [89–91] discuss the scalability issues of the distributed presence server architecture. Saint Andre [89] analyzes the traffic generated as a result of presence information between users of inter-domains that support the XMPP. Houri *et al.* [90] show that the amount of presence traffic in SIMPLE [83] can be extremely heavy, and they analyze the effect of a large presence system on the memory and CPU loading. Those works in [91,92] study related problems and developing an initial set of guidelines for optimizing inter-domain presence traffic and present a DHT-based presence server architecture.

Recently, presence services are also integrated into mobile services. For example, 3GPP has defined the integration of presence service into its specification in UMTS. It is based on SIP [93] protocol, and uses SIMPLE [85] to manage presence information. Recently, some mobile devices also support mobile presence services. For example, the Instant Messaging and Presence Services (IMPS) was developed by the Wireless Village consortium and was united into Open Mobile Alliance (OMA) IMPS [94] in 2005. In [95], Chen *et al.* proposed a weakly consistent scheme to reduce the number of updating messages in mobile presence services of IP Multimedia Subsystem (IMS). However, it also suffers scalability problem since it uses a central SIP server to perform presence update of mobile users [96]. In [97], authors presented the server scalability and distributed management issues in IMS-based presence services.

## 4.7 Conclusion

In this chapter, we have presented PresenceCloud, a scalable server architecture that supports mobile presence services in large-scale social network services. We have shown that PresenceCloud achieves low search latency and enhances the performance of mobile presence services. In addition, we discussed the scalability problem in server architecture designs, and introduced the buddy-list search problem, which is a scalability problem in the distributed server architecture of mobile presence services. Through a simple mathematical model, we show that the total number of buddy search messages increases substantially with the user arrival rate and the number of presence servers. The results of simulations demonstrate that PresenceCloud achieves major performance gains in terms of the search cost and search satisfaction. Overall, PresenceCloud is shown to be a scalable mobile presence service in large-scale social network services.

# CHAPTER 5
# Conclusion

"*No one fits all.*"
–Unknown

THE main research focus of this dissertation is to develop and to evaluate system-level architectures and overlay techniques to support information dissemination services in publish and subscribe paradigms. In this dissertation, we focus on understanding and investigating the following questions **Q1)** *How to rapidly disseminate a large-sized file to many subscribers in the Internet when the publisher releases the file*? **Q2)** *What is the relation between a single overlay tree with a fixed uplink rate and the broadcast operation itself, and how to construct a single overlay tree that minimize the maximum completion time in heterogeneous networks*? **Q3)** *Does there exist any other messages dissemination service that significantly outperforms those based distributed hash tables (DHT) both on scalability and efficiency for mutable and dynamic information, such as mobile presence messages?* To address these questions, we propose solutions for these questions as follows.

In CHAPTER 2, we address the Q1. We present the Bee protocol, which is a best-effort peer-to-peer data dissemination protocol aiming at minimizing the maximum dissemination time for all peers to obtain the complete file. Bee is a decentralized protocol that organizes peers into a randomized mesh-based overlay and each peer only works with local knowledge. We devise in Bee protocol a slowest peer first strategy to boost the speed of dissemination, and a topology adaptation algorithm that adapts the number of connections based on upload bandwidth capacity of a peer. Moreover, Bee is designed to support network heterogeneity and deal with the flash crowd arrival pattern without sacrificing the dissemination speed. We show that its performance can approach lower bound of the maximum dissemination time based on experimental results. The results also show that the Bee protocol does not sacrifice much fairness in terms of the amount of upload data

and dissemination.

In CHAPTER 3, we focus on the Q2. We introduce a novel LockStep Broadcast Tree (LSBT) to model the Big Data Broadcasting problem. We present a polynomial-time algorithm to build an optimal LSBT. The main idea in our LSBT model is to define a basic unit of upload bandwidth, $r$, such that each node uses only integer multiples of $r$ in broadcasting. In so doing, the number of uplinks of a node is proportion to the capacity of the node. Furthermore, we also divide the broadcasting data into $m$ chunks. These chunks are then broadcast down the tree by the nodes in a pipeline manner. We show that in LSBT model, the maximum number of rounds required to complete the broadcast of entire data chunks is O($m + logn$), where $n$ is the number of nodes. In a homogeneous network environment in which each node has the same uploading capacity $c$, we show that the optimal uplink rate $r^*$ of LSBT is either $c/2$ or $c/3$ . For heterogeneous environments, we present an O($n \log^2 n$) algorithm to select the optimal uplink rate $r^*$ and to construct the optimal LSBT. Numerical results show that the maximum completion time of our LSBT approximates to the optimum of the big data broadcast problem.

In CHAPTER 4, we focus on investigating the Q3. We propose an efficient and scalable server-overlay architecture, called PresenceCloud, which enables mobile presence services to support large-scale social network applications. When a mobile user joins a network, PresenceCloud searches for the presence of his/her friends and notifies them of his/her arrival. PresenceCloud organizes presence servers into a quorum-based server-to-server architecture for efficient presence searching. It also leverages a directed search algorithm and a one-hop caching strategy to achieve small constant search latency. We analyze the performance of PresenceCloud in terms of the search cost and search satisfaction level. The search cost is defined as the total number of messages generated by the presence server when a user arrives; and search satisfaction level is defined as the time it takes to search for the arriving user's friend list. The results of simulations demonstrate that PresenceCloud achieves performance gains in the search cost without compromising search satisfaction.

In summary, with the rapid advances of ICT and the increasing popularity of mobile devices, we witness the continuous escalation of User Generated Content (UGC) applications and the enormous data sets generated from users and machines. The imple-

mentations of these applications usually involve the information dissemination between publishers and multiple subscribers. Traditional client/server information dissemination architecture can hardly solve the problem because the server-side become the bottleneck as the whole system scales. Moreover, the world is becoming more flat, and this increasingly flat world entails more information availability and interchanges. Hence, the truth is that there is *no one fit all solution* for information dissemination services, future information dissemination services of publish and subscribe paradigms shall call for a generic, resilient, efficient and reliable platform for the demand of **everyone** in the Internet.

As a possible direction for future work, we are studying the possibility to leverage overlay systems to support information dissemination in disaster management. As part of our ongoing research, we are studying how to leverage the principles of UGC and the publish and subscribe paradigm to design a efficient, reliable and resilient overlay system for disaster information dissemination.

# Bibliography


[1] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[2] B. Cohen, "Incentives build robustness in bittorrent," *Proc. of ACM P2PECON*, 2003.

[3] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A cooperative bulk data transfer protocol," *Proc. of IEEE INFOCOM*, 2004.

[4] D. Kosti, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," *Proc. of ACM SOSP*, 2003.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in a cooperative environment," *Proc. of ACM SOSP*, 2003.

[6] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and S. Mehrotra, "Crew: A gossip-based flash-dissemination system," *Proc. of IEEE ICDCS*, 2006.

[7] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy, "Transport layer identification of p2p traffic," *Proc. of ACM IMC*, 2004.

[8] R. E. Bryant, R. H. Katz, and E. D. Lazowska, "Big-data computing: Creating revolutionary break throughs in commerce, science, and society," *In Computing Research Initiatives for the 21st Century.*, 2008.

[9] A. Szalay and J. Gray, "2020 computing: Science in an exponential world," *Nature 440, 413-414, March*, 2006.

[10] U. Rencuzogullari and S. Dwarkadas, "Dynamic adaptation to available resources for parallel computing in an autonomous network of workstations," *Proc. of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2001.

[11] "Facebook, http://www.facebook.com."

[12] S. Bhattacharyya, J. F. Kurose, D. F. Towsley, and R. Nagarajan, "Efficient rate-controlled bulk data transfer using multiple multicast groups," *IEEE/ACM Trans. Netw. (TON)*, 2003.

[13] X. Yang and G. de Veciana, "Service capacity of peer to peer networks," *Proc. of IEEE INFOCOM*, 2004.

[14] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," *Proc. of ACM SIGCOMM*, 2004.

[15] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "A performance study of bittorrent-like peer-to-peer systems," *IEEE JSAC*, 2007.

[16] R. Kumar and K. Ross, "Peer assisted file distribution: The minimum distribution time," *Proc. of IEEE Workshop on Hot Topics in Web Systems and Technologies*, 2006.

[17] B. Fan, J. C. Lui, and D. Chiu, "The delicate tradeoffs in designing bittorrent-like file sharing protocols," *To appear in IEEE/ACM Trans. Netw.*, 2008.

[18] Y.-M. Chiu and D. Y. Eun, "Minimizing file download time in stochastic peer-to-peer networks," *To appear in IEEE/ACM Trans. Netw.*, 2008.

[19] A. Bharambe, C. Herley, and V. Padmanabhan, "Analyzing and improving bittorrent performance," *Proc. of IEEE INFOCOM*, 2006.

[20] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," *Proc. of the ACM IMC*, 2003.

[21] A. Akella, S. Seshan, and A. Shaikh, "An empirical evaluation of wide-area internet bottlenecks," *Proc. of ACM IMC*, 2003.

[22] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in bittorrent systems," *in Proc. of ACM SIGMETRICS*, 2007.

[23] S. Saroiu, P. K. Gummadi, and S. D, "A measurement study of peer-to-peer file sharing systems," *Proc. of ACM MMCN*, 2002.

[24] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice, "Dissecting bittorrent: Five months in a torrent.s lifetime," *Proc. of PAM*, 2004.

[25] X. Zheng, C. Cho, and Y. Xia, "Optimal peer-to-peer technique for massive content distribution," *Proc. of IEEE INFOCOM*, 2008.

[26] C.-J. Wu, C.-Y. Li, and J.-M. Ho, "Improving the download time of bittorrent-like systems," *Proc. of IEEE ICC*, 2007.

[27] A. Papadimitriou and A. Delis, "Flash data dissemination in unstructured peer-to-peer networks," *Proc. of the IEEE ICPP*, 2008.

[28] "Akamai technologies, inc." *http://www.akamai.com/*.

[29] K. Park and V. S. Pai, "Scale and performance in the coblitz large-file distribution service," *Proc. of USENIX NSDI*, 2006.

[30] G. Brumfiel, "High-energy physics: Down the petabyte highway," *Nature 469, 282-283 January*, 2011.

[31] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Proc. of OSDI*, 2004.

[32] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, , and R. E. Gruber, "Bigtable: A distributed storage system

for structured data," *Proc. of OSDI*, 2006.

[33] W. D. Hillis and G. L. Steele, Jr., "Data parallel algorithms," *Communications of the ACM*, vol. 29, pp. 1170–1183, December 1986.

[34] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *Proc. of ACM SIGCOMM*, pp. 98–109, 2011.

[35] D. Nukarapu, B. Tang, L. Wang, and S. Lu, "Data replication in data intensive scientific applications with performance guarantee," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1299 –1306, aug. 2011.

[36] A. Epstein, D. H. Lorenz, E. Silvera, and I. Shapira, "Virtual appliance content distribution for a global infrastructure cloud service," *Proc. of IEEE INFOCOM*, pp. 516–524, 2010.

[37] C. Peng, M. Kim, Z. Zhang, and H. Lei, "Vdn: Virtual machine image distribution network for cloud data centers," *Proc. of IEEE INFOCOM*, 2012.

[38] S. Khuller and Y.-A. Kim, "Broadcasting in heterogeneous networks," *Algorithmica*, vol. 48, no. 1, Mar. 2007.

[39] J. Mundinger, R. Weber, and G. Weiss, "Optimal scheduling of peer-to-peer file dissemination," *Journal of Scheduling*, vol. 11, no. 2, 2008.

[40] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "P2p streaming capacity under node degree bound," *Proc. of IEEE INFOCOM*, 2007.

[41] O. Beaumont, L. Eyraud-Dubois, and S. K. Agrawal, "Broadcasting on large scale heterogeneous platforms under the bounded multi-port model," *Proc. of IEEE IPDPS*, 2011.

[42] S. M. Hedetniemi, S. T. Hedetniemi, and A. Liestman, "A survey of gossiping and broadcasting in communication networks," *Networks*, 1988.

[43] P. Liu, "Broadcast scheduling optimization for heterogeneous cluster systems," *J. Algorithms*, vol. 42, no. 1, Jan. 2002.

[44] K. Wang, J. Li, and L. Pan, "Fast file dissemination in peer-to-peer networks with upstream bandwidth constraint," *Future Generation Computer Systems*, vol. 26, July 2010.

[45] K.-S. Goetzmann, T. Harks, M. Klimm, and K. Miller, "Optimal file distribution in peer-to-peer networks," *Proc. of ISAAC*, 2011.

[46] M. Deshpande, N. Venkatasubramanian, and S. Mehrotra, "Heuristics for flash-dissemination in heterogenous networks," *Proc. of the 13th international conference on High Performance Computing*, 2006.

[47] C.-J. Wu, C.-Y. Li, K.-H. Yang, J.-M. Ho, and M.-S. Chen, "Time-critical data

dissemination in cooperative peer-to-peer systems," *Proc. of IEEE GLOBECOM*, 2009.

[48] A. Passarella, "A survey on content-centric technologies for the current internet: Cdn and p2p solutions," *Computer Communications*, 2012.

[49] M. A. Brown, "Traffic control howto. chapter 6. classless queuing disciplines," *http://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html*, 2006.

[50] A. Cayley, "A theorem on trees," *Quarterly Journal of Mathematics*, vol. 23, 1889.

[51] R. Thommes and M. Coates, "Bittorrent fairness: Analysis and improvements," *Proc. of Workshop Internet, Telecom. and Signal Proc.*, December 2005.

[52] Murder, "https://github.com/lg/murder."

[53] S. ul Islam, K. Stamos, J.-M. Pierson, and A. Vakali, "Utilization-aware redirection policy in cdn: A case for energy conservation," *Proc. of Information and Communication on Technology for the Fight against Global Warming*, 2011.

[54] J. Edmonds, "Edge-disjoint branchings, in combinatorial algorithms," *Algorithmics Press*, 1972.

[55] G. M. Ezovski, A. Tang, and L. L. H. Andrew, "Minimizing average finish time in p2p networks," *Proc. of IEEE INFOCOM*, 2009.

[56] C. Chang, T. Ho, M. Effros, M. Medard, and B. Leong, "Issues in peer-to-peer networking: a coding optimization approach," *Proc. of IEEE International Symposium on Network Coding (NetCod)*, 2010.

[57] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance bounds for peer-assisted live streaming," *Proc. of ACM SIGMETRICS*, 2008.

[58] S. Liu, M. Chen, S. Sengupta, M. Chiang, J. Li, and P. A. Chou, "P2p streaming capacity under node degree bound," *Proc. of IEEE ICDCS*, 2010.

[59] "Twitter, http://twitter.com."

[60] "Google latitude, http://www.google.com/intl/enus/latitude/intro.html."

[61] "buddycloud, http://buddycloud.com/."

[62] "Mobile instant messaging," *http://en.wikipedia.org/wiki/Mobile_instant_messaging*.

[63] R. B. Jennings, E. M. Nahum, D. P. Olshefski, D. Saha, Z.-Y. Shae, and C. Waters, "A study of internet instant messaging and chat protocols," *IEEE Network*, 2006.

[64] A. Houri, E. Aoki, S. Parameswar, T. Rang, , V. Singh, and H. Schulzrinne, "Presence interdomain scaling analysis for sip/simple," *RFC Internet-Draft*, 2009.

[65] M. Maekawa, "A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems," *ACM Transactions on Computer Systems*, 1985.

[66] D. Eastlake and P. Jones, "Us secure hash algorithm 1 (sha1)," *RFC 3174*, 2001.

[67] M. Steiner, T. En-Najjary, and E. W. Biersack, "Long term study of peer behavior in the kad dht," *IEEE/ACM Trans. Netw.*, vol. 17, pp. 1371–1384, October 2009.

[68] K. Singh and H. Schulzrinne, "Failover and load sharing in sip telephony," *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, July 2005.

[69] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet," *IEEE/ACM Tran. on Networking*, 2003.

[70] X. Chen, S. Ren, H. Wang, and X. Zhang, "Scope: scalable consistency maintenance in structured p2p systems," *Proc. of IEEE INFOCOM*, 2005.

[71] K. P. Gummadi, S. Saroiu, and S. D. Gribble., "King: Estimating latency between arbitrary internet end hosts," *Proc. of ACM IMW*, 2002.

[72] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," *Proc of ACM MASCOTS*, 2001.

[73] R. Cox, A. Muthitacharoen, and R. T. Morris, "Serving dns using a peer-to-peer lookup service," *Proc. of IPTPS*, 2002.

[74] V. Ramasubramanian and E. G. Sirer, "Beehive: 0(1) lookup performance for power-law query distributions in peer-to-peer overlays," *Proc. of USENIX NSDI*, 2004.

[75] A. Abdul-Rahman and S. Hailes., "A distributed trust model," *Proc. of the workshop on New security paradigms*, 1997.

[76] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei, "Quantifying skype user satisfaction," *Proceedings of ACM SIGCOMM*, 2006.

[77] P. Anick, "Using terminological feedback for web search refinement: a log-based study," *Proceedings of ACM SIGIR conference on Research and development in informaion retrieval*, pp. 88–95, 2003.

[78] "Gobalindex, http://www.skype.com/intl/en-us/support/user-guides/p2pexplained/."

[79] Z. Xiao, L. Guo, and J. Tracey, "Understanding instant messaging traffic characteristics," *Proc. of IEEE ICDCS*, 2007.

[80] C. Chi, R. Hao, D. Wang, and Z.-Z. Cao, "Ims presence server: Traffic analysis and performance modelling," *Proc. of IEEE ICNP*, 2008.

[81] "Instant messaging and presence protocol ietf working group. http://www.ietf.org/html.charters/impp-charter.html."

[82] "Extensible messaging and presence protocol ietf working group. http://www.ietf.org/html.charters/xmpp-charter.html."

[83] "Sip for instant messaging and presence leveraging extensions ietf working group.

http://www.ietf.org/html.charters/simple-charter.html."

[84] P. Saint-Andre., "Extensible messaging and presence protocol (xmpp): Instant messaging and presence describes instant messaging (im), the most common application of xmpp," *RFC 3921*, 2004.

[85] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle, "Session initiation protocol (sip) extension for instant messaging," *RFC 3428*, 2002.

[86] "http://www.jabber.org/."

[87] "Peer-to-peer session initiation protocol ietf working group. http://www.ietf.org/html.charters/p2psip-charter.html."

[88] K. Singh and H. Schulzrinne, "Peer-to-peer internet telephony using sip," *Proc. of ACM NOSSDVA*, 2005.

[89] P. Saint-Andre, "Interdomain presence scaling analysis for the extensible messaging and presence protocol (xmpp)," *RFC Internet Draft*, 2008.

[90] A. Houri, T. Rang, and E. Aoki, "Problem statement for sip/simple," *RFC Internet-Draft*, 2009.

[91] A. Houri, S. Parameswar, E. Aoki, V. Singh, and H. Schulzrinne, "Scaling requirements for presence in sip/simple," *RFC Internet-Draft*, 2009.

[92] S. A. Baset, G. Gupta, and H. Schulzrinne, "Openvoip: An open peer-to-peer voip and im system," *Proc. of ACM SIGCOMM*, 2008.

[93] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: Session initiation protocol," *RFC 3261*, 2002.

[94] O. M. Alliance, "Oma instant messaging and presence service," 2005.

[95] W.-E. Chen, Y.-B. Lin, and R.-H. Liou, "A weakly consistent scheme for ims presence service," *IEEE Transactions on Wireless Communications*, 2009.

[96] N. Banerjee, A. Acharya, and S. K. Das, "Seamless sip-based mobility for multimedia applications." *IEEE Network*, vol. 20, no. 2, pp. 6–13, 2006.

[97] P. Bellavista, A. Corradi, and L. Foschini, "Ims-based presence service with enhanced scalability and guaranteed qos for interdomain enterprise mobility," *IEEE Wireless Communications*, 2009.