PAC'nPost: A Peer-to-Peer Micro-blogging Social Network

Harshvardhan Asthana

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University College London.

Department of Computer Science

University College London

July 27, 2014

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

Harshvardhan Asthana

To my parents

for all their love and encouragement over the years

Abstract

In this thesis we provide a framework for a micro-blogging social network in an unstructured peer-to-peer network. At a user level, a micro-blogging service provides (i) a means for publishing a micro-blog, (ii) a means to follow a micro-blogger, and (iii) a means to search for micro-blogs containing keywords. Since unstructured peer-to-peer networks do not bind either the data or the location of data to the nodes in the network, search in an unstructured network is necessarily probabilistic. Using the probably approximately correct (PAC) search architecture, the search of an unstructured network is based on a probabilistic search that queries a fixed number of nodes. We provide a mechanism by which information whose creation rate is high, such as micro-blogs, can be disseminated in the network in a rapid-yet-restrained manner, in order to be retrieved with high probability. We subject our framework to spikes in the data creation rate, as is common in micro-blogging social networks, and to various types of churn.

Since both dissemination and retrieval incur bandwidth costs, we investigate the optimal replication of data, in the sense of minimizing the overall system bandwidth. We explore whether replicating the micro-blog posts of users with a larger number of followers more than the posts of other users can reduce the overall system bandwidth.

Finally, we investigate trending keywords in our framework. Trending keywords are important in a micro-blogging social network as they provide users with breaking news they might not get from the users they follow. Whereas identifying trending keywords in a centrally managed system is relatively straightforward, in a distributed decentralized system, the nodes do not have access to the global statistics such as the frequency of the keywords and the information creation rate. We describe a two-step algorithm which is capable of detecting multiple trending keywords with moderate increase in bandwidth.

Acknowledgements

I am deeply indebted to my supervisor, Prof. Ingemar Cox, for all his advice and guidance. I have learnt more in the last few years than in any point in my life. Many thanks to Dr. Jun Wang, my second supervisor, for all his help. Pursuing a PhD has been a wonderful experience and I will remain forever grateful for the opportunity.

I would thank Dr. Mirco Musolesi and Dr. Miguel Rio for examining this work. The discussion in the examination has given me many new ideas for future work.

Many thanks to the whole MediaFutures team. It has been great fun working with you folks. None of this would have been possible without the support of my family, and I would like to thank each and every one of you for your support and kind words.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last decade, the internet has been witnessing a paradigm shift with the emergence of collaborative sites and online social networks. Increasingly, users are becoming publishers of information rather than passive consumers. With the launch of Twitter in 2006, the internet saw an explosion in the popularity of micro-blogging. Whereas regular blog postings may contain multiple paragraphs and are usually published once a week or at a lower frequency, the short nature (typically SMS sized message) of a micro-blog encourages users to publish multiple micro-blog posts each day, usually consisting of personal status updates, current events, news etc [JSFT07, KLPM10].

Twitter allows users to *follow* other users. When a user posts a tweet (a micro-blog post), it is displayed to all of his or her followers. Apart from the ability to follow individuals and perform keyword search across all publicly posted tweets, Twitter also displays *trending* keywords. This alerts users of breaking news which they may not get via individuals they follow, and invites them to perform a search on the trending keywords. An indication that Twitter is the best platform for breaking news is that it is the choice of medium used by journalists. Twitter allows journalists to build up a readership via their followers, to whom they tweet breaking news. Since its launch, Twitter has built up a user base of 200 million users who post approximately 400 million tweets per day [Tel13].

Whilst web search, online social networking, and micro-blogging have become immensely popular, there are a variety of concerns pertaining to these services, such as:

- **Monopolies**: A few select companies have become the primary providers of these technologies. In the western world and much of Asia, Google and Microsoft account for most web searches, Facebook is the dominant social networking site, and Twitter monopolizes micro-blogging. In China, Renren and Weibo are the dominant social networking and micro-blogging platforms respectively. Together, these companies spend billions of dollars to support their data centre facilities. This high cost may prohibit new challengers and may impede innovations. Another concern is that these web giants may abuse their dominance by censoring or demoting within the search results, information critical of themselves. More likely, these web giants could manipulate search results to favour sites which are commercially beneficial for themselves, rather than ones which are objectively more relevant to the search query. For instance, Google has announced that, for certain product searches, it will not maintain a strict separation between search results and advertising [Goo12].

- **Ownership of data**: Users have been concerned about the legal ownership of the text and multimedia posted on social networking sites and their private information being used for commercial gain. This issue of data ownership has motivated peer-to-peer alternatives to Facebook such as Diaspora[1].

- **Central point of failure**: Another concern with a centrally managed service is that outages can render it unavailable to all its users and Twitter has faced several of these outages since its launch (Figure 1.1).

- **Censorship**: Since a micro-blogging social network provides a convenient way to report and spread news and opinion, providers can become target of censorship by authoritarian regimes. For example, Twitter is banned in China [BOS12]. The Chinese micro-blogging website Weibo, which has over 500 million registered users, and other social networking sites users are monitored by an astonishing 2 million people [BBC13] and politically sensitive posts and images are flagged and removed [Pro13]. Users often have to resort to round-about ways to evade censors, for example posting *17+1* instead of *18* for the 18th national party congress [BBC12a]. Twitter itself has announced its own intention of censoring tweets by country [BBC12b].

- **Anonymity**: There are concerns about privacy and anonymity as data posted by individuals can be used to identify them or used to infer sensitive personal information. For example, Twitter has released information to law enforcement authorities about users who presumed they were blogging anonymously [BBC11]. Kosinski *et al.* [KSG13] were able to predict, with high probability, gender, political leanings, religion, ethnicity, and sexual orientation of users by mining their "likes" on Facebook.

Twitter is over capacity.

Please wait a moment and try again. For more information, check out **Twitter Status.**

© 2012 Twitter About Help Status

Figure 1.1: The infamous Twitter *fail whale*.

Such concerns motivate the case for a P2P alternative. A distributed decentralised system would be much harder to censor or block, and by default would have no single point of failure. Unlike centrally managed social networks such as Facebook, Twitter, and Google+ where users have to reveal personal information in order to participate, a P2P system need not ask for any personal information from its users.

Peer-to-peer systems connect users so they can communicate and share content directly by eliminating centralized servers. In a P2P system, each participating node acts as both server and client. Each participating node shares a fraction of its resources, such as processing power, storage, and bandwidth. P2P systems offer a low cost of entry and are used to run a variety of applications such as internet

[1]`https://joindiaspora.com`

chat [GD05], file sharing and content delivery [PGES05], multimedia streaming [YJC07], as well as digital currencies such as BitCoin [Nak08].

P2P networks can be generally categorized into two classes, namely structured and unstructured networks. Structured networks, loosely defined as networks where nodes maintain routing information about how to reach all other nodes in the overlay, are typically based on distributed hash tables (DHTs). They bind data to designated locations within the network, and this binding allows any participating node to efficiently retrieve the data associated with a given key [BYL09]. However this binding also makes structured networks particularly susceptible to adversarial attack [SM02, UPS11], and to churn, i.e. the random entry and exit of nodes to the network. Furthermore, since the DHT needs to be kept up-to-date, structured networks are generally unsuitable for data which is highly dynamic, such as a micro-blogging social network. Finally, structured P2P networks are unsuitable for multi-term keyword search as demonstrated by a feasibility study by Li *et al.* [LLH$^+$03].

Unstructured networks, loosely defined as a networks in which a node relies only on its adjacent nodes for delivery of messages to other nodes in the overlay, generally have no binding between data and nodes [BYL09]. As such, they are much less affected by churn, and are generally more resistant to adversarial attack. However, since a particular document being sought by a user can be anywhere in the network, the only way to guarantee searching the entire collection is to exhaustively query all nodes in the network. This is, of course, impractical. To be practical, any search must only query a relatively small subset of nodes in the network. Thus, search in an unstructured P2P network is necessarily probabilistic.

This thesis looks at a peer-to-peer micro-blogging social network implemented on top of a unstructured P2P network. We utilize the probably approximately correct (PAC) search architecture [CFH09, CZFH10] for retrieval of published micro-blog posts with high probability. The PAC search architecture queries a fixed number of nodes, and a strong theoretical framework exists to predict the probability of a successful search based on the distribution of documents in the network.

Unlike recently proposed P2P social networks, such as Diaspora, the design of a distributed micro-blogging social network described here is not intended to address the issue of data ownership. Our focus is on data availability - published micro-blog posts should be available even if the publisher node is offline, and information retrieval (IR) - the ability of participating nodes to retrieve, with high probability, the micro-blog posts of peers that they follow as well as to be able to perform keyword search across all published posts, as akin to a search engine.

1.1 Contributions and Structure

This thesis provides the framework for a peer-to-peer micro-blogging social network based upon the PAC search architecture, which we call *PAC'nPost*. We utilize PAC's measure of correctness, known as *accuracy* to retrieve both the followed micro-blog posts and posts which match a keyword query.

In PAC, the probability of retrieving a document is a function of (i) the number of nodes queried, and (ii) the replication of the document, i.e. the number of nodes on which the document is stored. As we increase the number of nodes queried, the latency also increases, so it is desirable to query fewer nodes. For a required probability of retrieval, and a fixed number of nodes to be queried, i.e. a fixed latency, we analytically derive the minimum number of nodes a document must be stored on. The problem of information retrieval over an unstructured P2P network in which the frequency of information creation is high (such as a micro-blogging social network) then becomes how to replicate new information to this number of nodes given the constraints of (i) time (we wish to propagate the information as quickly as possible), and (ii) bandwidth (we wish to propagate the information using as little bandwidth as possible, otherwise our solution will not be scaleable).

In the PAC'nPost framework, a user publishes micro-blog posts on his or her local node which are replicated onto other randomly chosen nodes so that it can be retrieved with high probability by the user's *followers*. To retrieve the micro-blogs post a user is following, his or her node periodically queries randomly chosen peers. These replication and retrieval processes can occur simultaneously or separately, based on the configuration of PAC'nPost. To participate in PAC'nPost each node must contribute some disk space which is utilized to store micro-blog posts of other users as well as to index the posts for keyword search. The node is also required to make periodic queries to a predetermined number of random peers to facilitate the retrieval of micro-blog posts.

In order to disseminate micro-blog posts to the required number nodes so that they can be retrieved with high accuracy, we utilize a modified gossip protocol. Gossip protocols, also known as rumour spreading protocols, usually have the goal of ensuring that the rumour spreads to the entire network at the fastest possible rate. Our requirement is subtly different; we require that a document be replicated rapidly onto a fraction of the network but then no further. We call this *rapid-yet-restrained* dissemination. We subject our framework to heavy churn, and spikes in the data creation rate.

Previous research in peer-to-peer IR has generally assumed that the information on the network is generally static. Furthermore, in the study of gossip protocols, a rumour is not impeded by a rival concurrent rumour. In a micro-blogging social network multiple posts are created every second, and thus our framework is designed to facilitate *multiple simultaneous rumours*.

Since both dissemination and retrieval incur bandwidth costs, we investigate the optimal replication of data, in the sense of minimizing bandwidth, and the balance between the number of nodes a micro-blog post is replicated to and the number of nodes that must be queried in order to guarantee a specified retrieval accuracy. We examine whether replicating posts of users with large number of followers to more nodes can reduce the overall bandwidth of a system where nodes query periodically to retrieve their followed micro-blog posts.

We examine the retrieval of trending keywords in our framework. Whereas identifying trending keywords in a centralized system is relatively straightforward, in a decentralized distributed system, the nodes do not have access to global statistics such as the word frequencies and the data creation rate. We provide a 2-step method which can display trending keywords to nodes participating in the network.

We conducted simulations with post creation rates of 10 and 100 times Twitter's tweets-per-second in networks of 10,000 and 100,000 nodes to evaluate our framework. We chose micro-blog creation rates magnitudes higher than that of Twitter to demonstrate that our framework can facilitate high data rates without overwhelming any given node with a deluge of data processing. We emphasize that the framework described in this thesis works in a fully distributed manner and required no central coordination.

The remainder of the thesis is organized as follows:

- **Chapter 2 - Background**: first reviews prior work in information retrieval and peer-to-peer systems. Next, the PAC search architecture is described, and the chapter concludes by describing recently proposed P2P social networks.

- **Chapter 3 - The PAC'nPost Framework** [AC12, AC13b]: analyses the problem into its two main components: dissemination and retrieval. In order to facilitate dissemination of published micro-blog posts, nodes share a small number of posts as they regularly query for their own followed posts. These shared micro-blog posts form the node's *transfer buffer*. Two types of transfer buffers are examined - one in which a fixed number of posts are selected, and a flexible size buffer in which posts are selected into via a dissemination parameter.

- **Chapter 4 - Bandwidth Optimization** [AC13a]: investigates the optimal replication of data in order to minimize the overall system bandwidth. The theoretical probabilistic analysis predicts that micro-blog posts should be disseminated onto approximately 20% and 6% of nodes in networks of 10,000 and 100,000 nodes respectively. The analysis is then extended to the case of non-uniform replication where the most popular 1% of micro-blog posts are disseminated more than others.

- **Chapter 5 - Trending Keywords** [AC13d, AC13c]: explores trending keywords in a decentralized distributed system. A two-step algorithm is presented which can successfully retrieve trending keywords at the nodes in the network.

- **Chapter 6 - Conclusions**: proposes future research and concludes the thesis.

Chapter 2

Background

In this chapter we review the related prior work relevant to this thesis. In Section 2.1 we introduce basic information retrieval (IR) fundamentals. Next, in Section 2.2 we review peer-to-peer systems and relevant processes such as gossiping and search. We then detail probably approximately correct (PAC) search in Section 2.5. Finally we describe some recently proposed peer-to-peer social networks in Section 2.6.

2.1 Information Retrieval

Information retrieval (IR) is the area of study concerned with searching for documents, for information within documents, and for meta-data about documents within a defined collection [MRS08]. In it's simplest form, the user expresses his/her *information need* as a *query*, and an IR system returns the identified documents by matching the query with it's *inverted index*. A document is judged to be *relevant* if it satisfies the user's information need. Figure 2.1 displays the general model of IR.

For an IR system to return relevant documents to the user, it must maintain an inverted index of the document in a collection. The index is created by the processing of documents which generally consists of the following steps:

1. Lexical analysis of the text with the objective of processing digits, hyphens, punctuation marks, etc.

2. Elimination of *stop words* (e.g. "the", "and" etc.) with the objective of filtering out words with very low discrimination values for retrieval purposes.

3. Stemming [Har91] of words to remove prefixes and suffixes to allow the retrieval of documents containing variations of the query (e.g. run, runs, running).

2.1.1 Partitioning of the index

For large IR systems the inverted index becomes too big to fit on one server and has to be distributed among multiple machines. There are two main strategies for partitioning the index *term partitioning* and *document partitioning*. Figure 2.2 illustrates both strategies. Document partitioning splits the document collection in several sub-collections and each sub-collection is indexed locally and independently on a different server. Thus, document partitioning is sometimes called local index partitioning. A query

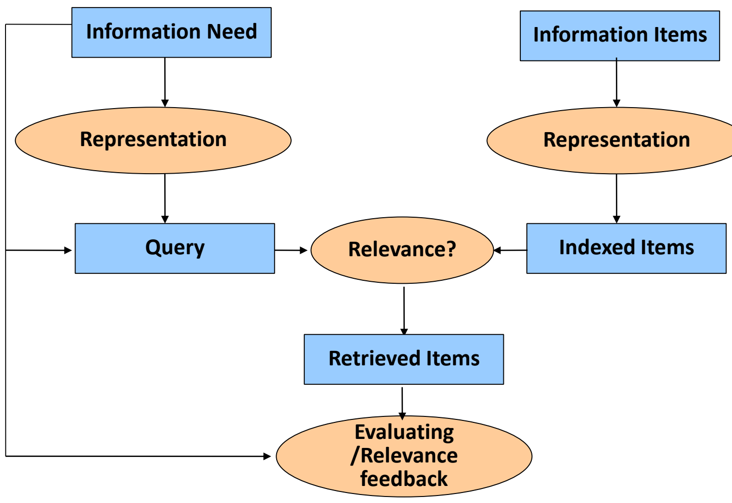


Figure 2.1: The general model of Information Retrieval

is processed by all servers in parallel and the final result is aggregated from the top-k local answers supplied by each server. Important advantages of document partitioning are the simplicity of the indexing procedure and nearly even load balancing between the servers. On the other hand, each query has to be processed by each server which increases the processing costs.

Term partitioning assigns terms to servers such that each server maintains complete posting lists for certain terms. Hence, term partitioning is sometimes called global partitioning. To process a query only the servers responsible for the query terms have to be contacted. However indexing is costly and it is hard to balance the load as the term frequency distribution follows a power law. Intersecting posting lists that are stored on different servers can be time and bandwidth consuming. Nevertheless, P2P IR approaches built on top of structured P2P networks often choose term partitioning [BMT$^+$05].

2.1.2 Retrieval Models

In order to find the relevant documents for a given query, the IR system must employ a retrieval model to score and rank the documents for display to the user. Generally, retrieval models can be classified into three categories (in ascending order of complexity):

1. Boolean models
2. Vector Space models
3. Probabilistic models

The Boolean model is a simple retrieval model based on set theory and Boolean algebra. The Boolean models predict whether each document is either relevant or non-relevant based on the existence of query terms within a document. There is no notion of a partial match with the query.

To improve upon the Boolean model, other statistics can be utilized, such as the *term frequency* (tf), t_{ij}, which indicates how many times term j occurs in document i, and *document frequency* (df), a_j, which indicates the number of documents in the entire collection, which contain term j. Finally, the

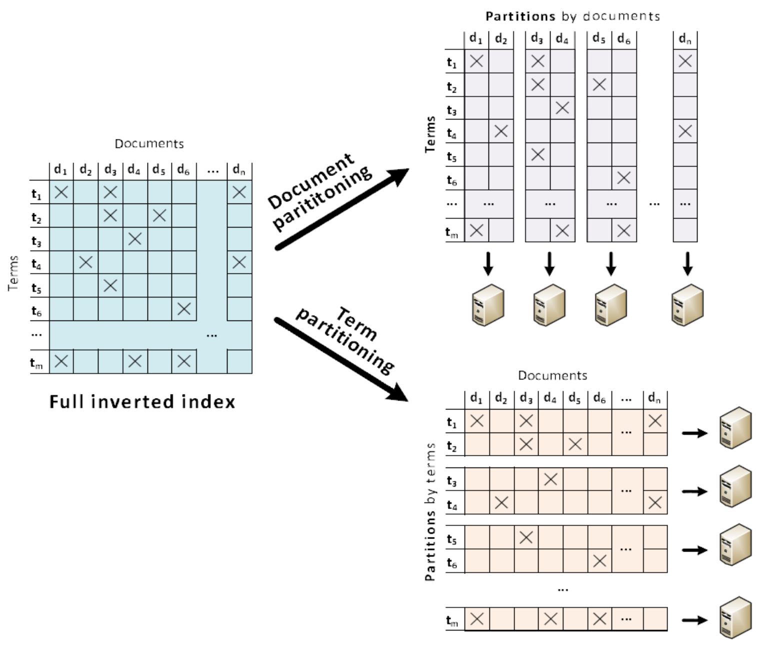


Figure 2.2: Document partitioning and Term partitioning of a large index onto multiple servers

inverse document frequency (IDF) is computed as:

$$\text{idf}_j = \log \frac{N - a_j + 0.5}{a_j + 0.5}$$

where N is the number of documents in the collection. The vector space model [SWY75] assigns a compound weight to each term using the term frequency (t_{ij}) and inverse document frequency (idf_j). Thus each query and document can be viewed as vector, and the documents can be ranked according to the cosine similarity:

$$Sim(q, d_i) = \frac{\overrightarrow{V}(q) \cdot \overrightarrow{V}(d_i)}{|\overrightarrow{V}(q)||\overrightarrow{V}(d_i)|}$$

where $\overrightarrow{V}(q)$ and $\overrightarrow{V}(d_i)$ are the vectors for query q and document d_i respectively.

Finally, Probabilistic models, introduced by Robertson *et al.* [RJ76] are based on the following assumption: Given a query q, the model assigns to each document d, as a measure of its similarity, the ratio P(d relevant-to q) / P(d not-relevant-to q), which computes the likelihood of the document d being relevant to q. One of the most widely used probabilist mode is BM25 [RWJ$^+$96]. Given a query q containing terms $q_1, \ldots, q_m$, the BM25 score for document i is:

$$\text{score}(q, d_i) = \sum_{j \in q} \text{idf}_j \cdot \frac{t_{ij} \cdot (k+1)}{t_{ij} + k \cdot (1 - b + b \cdot \frac{l_i}{L})}$$

where l_i is the length of document d_i, L is the average document length in the collection, and b and k are parameters usually set to 0.75 and 2.0 respectively.

2.1.3 Evaluation

In order to compare different IR systems, we need to judge their effectiveness. Generally, this is usually measured by two key metrics: *precision* and *recall*. Precision is the fraction of the documents retrieved by the system which are judged to be relevant:

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Recall is the fraction of the relevant documents which are retrieved by the system:

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

2.2 Peer-to-Peer Networks

P2P networks are decentralised distributed systems and enable computers to share and integrate their computing resources, data and services. In a P2P network, each node acts as both client and server, requesting resources from as well as routing queries and serving resources for other peer nodes [SYBA10]. A P2P network can be best defined as a logical overlay network over a physical infrastructure.

P2P networks can be classified in two ways: by degree of network centralization and by structure. We briefly describe these two classifications.

2.2.1 P2P Networks - Network Centralization

There are three types of P2P networks when categorized by network centralization: *centralized* P2P networks, *decentralized* P2P networks, and *hybrid* P2P networks.

In a centralized P2P system a centrally managed server keeps track of the meta-data about shared content. Peers connect to this server to locate their required files, and connect with the appropriate peers. Napster and BitTorrent (prior to version 4.2) were examples of such a service. The obvious drawback of centralized P2P networks is a potential single point of failure, which was a significant factor in Napster's downfall.

Decentralized P2P networks do not have any central server which stores the location of files in the network, and therefore have no single point of failure. The Gnutella network is one such decentralized system. In the Gnutella network, each peer node uses a Breadth-First Search to search the network by broadcasting the query with a Time-to-Live (TTL) to all connected peers, where TTL is the number of times a query can be forwarded before it is discarded. Each node which receives the query responds to the query if it finds the requested file in its own storage. Otherwise, the node decreased the value of the TTL and forwards the query to all its neighbours. This method of searching a network is known as *flooding*. The obvious drawback of this method is that it can generate a very large amount of huge network traffic.

Hybrid P2P networks, such as Gnutella2, overcome the drawbacks of centralized and decentralized P2P networks by classifying the nodes into a two-level hierarchy of super-nodes and leaves. High capacity reliable nodes are chosen as super-nodes and carry most of the burden in the network. Typically a super-node is connected to a few leaves, and the super-node maintains an index of all the files shared by the leaf nodes. The super-nodes also maintain connections with other super-nodes in the network. When a node requests a file, it sends a request to its super-node. The super-node checks its own index for the file and also forwards the request to other super-nodes, who check their index for the requested file. If the file is found, the file is transferred directly between the node which queried for the file and the nodes which have chosen to share the file.

2.2.2 P2P Networks - Network Structure

Another way of categorizing P2P networks is into *structured* and *unstructured* networks. Structured networks, loosely defined as networks where nodes maintain routing information about how to reach all other nodes in the overlay, are typically based on distributed hash tables (DHTs), bind data to designated locations within the network. DHTs are algorithms for efficient resource location in a distributed system. In a DHT-based network, each node is allocated a unique identifier in a numeric key space. The DHT also provides for *put*, *get*, and *delete* interfaces to insert, retrieve, and remove key/value pairs from the distributed system respectively. The network structure is sorted by routing tables stored on individual nodes. Each node only needs a small amount of routing information about other nodes.

Chord [SMK$^+$01], Pastry [RD01], Content-Addressable Network (CAN) [RFH$^+$01], and Kademlia [MM02] are examples of popular DHT-based structured networks. We refer the reader to [SYBA10] for an extensive overview of structured networks.

Unstructured networks, loosely defined as a networks in which a node relies only on its adjacent nodes for delivery of messages to other nodes in the overlay, generally have no binding between data and nodes. A common search method in unstructured networks is *blind search*.

One way to implement a blind search is via a *random walker* [LCC$^+$02, ALPH01], which aims to reduce the network traffic generated by flooding. Using random walkers, a node forwards the query to a randomly chosen neighbour rather than all its neighbours. Each node repeats this process until the requested file is found. This process can be made more efficient with the replication of files in the network. It can be shown that replicating files in proportion to the square-root of the normalized frequency of the query pertaining to the file minimizes the expected search length [LCC$^+$02, CS02]. Another blind search method is iterative deepening [YGM02] in which a querying node utilizes successive Breadth-First Searches queries with increasing depths. The query is terminated if the file is found or the maximum depth is reached.

2.3 Gossip Protocols

Gossip protocols, also known as rumour spreading protocols, provide an efficient way to rapidly spread information within a network. The theory of rumour spreading algorithms originates from the mathematical modelling of the spread of infectious diseases within an community [Bai75].

Suppose we have a group of n individuals, and at $t = 0$, only one individual knows the rumour. At time t, let x denote the individuals who do not know the rumour (*susceptibles*), and y denote the number of individuals who know the rumour (*infectives*), so that $x + y = n$. As presented in [Bai75], the number of individuals who have received the rumour, y, is given by

$$y = \frac{n}{1 + ne^{-nt\nu}}$$

where ν is the contact rate. The contact rate is analogous to the number of nodes each peer communicates with. Clearly, as $t \to \infty$, $y \to n$ and the entire network is infected.

One of the earliest uses of rumour spreading was by Demers *et al.* [DGH$^+$87] to synchronize replicated databases. Demers *et al.* introduce a new class of nodes which know the rumour but do not participate in spreading it (*stiflers*). An *infective* becomes a *stifler* with a probability ϕ when it is contacted by another *infective*. If we now set x (*susceptibles*), y (*infectives*), and z (*stiflers*) to be the fraction of the network such that $x + y + z = 1$. The rate of occurrence of new infections is proportional to the *infectives* and the *susceptibles*. Therefore the new infections in time Δt is $\nu xy\Delta t$ and the new removals (nodes turning into *stiflers*) is $\nu\phi y\Delta t$. The differential equations for infectives and susceptibles are

$$\frac{dx}{dt} = -\nu xy \qquad \frac{dy}{dt} = \nu xy - \nu\phi y$$

Combining the two equations and integrating, we can get the *infectives* expressed as a function of the *susceptibles* as:

$$y(x) = (1 + \frac{1}{\phi})(1 - x) + \phi\log(x)$$

$y(x)$ is zero (i.e. the network reaches a steady state) when

$$x = \exp\left(-(\frac{1}{\phi} + 1)(1 - x)\right)$$

For a ϕ value of 1 and 0.5, the rumour spreads to 80%, and 94% of the network respectively at the final steady state.

It is important to note that this describes gossip spreading in a fully connected graph, and that network structure is not taken into account. PAC (Section 2.5) also describes the network at a higher level than the network topology. PAC could be considered as a fully connected graph overlay where only z random connection are active when a node makes a request and messages travel directly to the target node. Of course the underlying network topology (e.g. homogeneous, random graph, power law) determines how many *actual* nodes the message must pass through until it reaches the target node, and consequently the time delay. We refer the reader to [MNP04, NMBM07, New02] for an extensive overview of information dissemination in various types of networks.

Apart from utilizing gossip spreading algorithms for information dissemination [DGH$^+$87, EGKM04], they can also be used for aggregation [KDG03, JMB05], as well as network management [VVS03, VGVS05]. The study of information diffusion is not limited to computer networks; it has been studied in the context of human social activity [ACD$^+$09], P2P recommendation [PGW$^+$08] as well as for viral marketing [Dom05, LAH07].

2.4 Probabilistic Search

Probabilistic storage and search in an unstructured P2P network can be modelled as follows. Given a set of n nodes in the network, we assume that the object of interest is stored on a random subset of r nodes. A query is issued to a random subset of z nodes. We are interested in the probability that the two subsets have a non-empty intersection, as this implies a successful search for that object. Information retrieval is broader than this, as the index, and associated queries, contain terms present not just in the file name, but also terms present within the file (document) itself. As such, matching of queries to documents is more ambiguous, and it is therefore necessary to provide a set of documents, usually ranked by relevance. Nevertheless, this model is appropriate for the class of information retrieval problems referred to as *known-item* search. And, the probabilistic model can be extended to encompass other information retrieval requirements, as discussed in Section 2.5.

2.4.1 Probabilistic Quorum Systems

Early work on probabilistic search in unstructured P2P networks has its origins in the study of probabilistic quorum systems [MRW97] to improve the availability and efficiency of replicated systems. A quorum system consists of a set of subsets of servers. Each subset is called a quorum and every pair of quorums intersect. This intersection property guarantees that any change caused by a write operation of one quorum can be detected by a read operation in any other quorum, and data consistency is thereby maintained. An inherent problem in a quorum system is that there is a trade-off between the load (access frequencies on servers) and the fault-tolerance (the fraction of servers that can fail simultaneously without impacting the system's normal operation) - improving one comes at the expense of the other. A *probabilistic* quorum system relaxes the strict data consistency constraint to be probabilistic. By so doing, higher fault-tolerance is achieved while maintaining the load level on the system.

A formal model of a probabilistic quorum system is as follows:

1. Denote the total set (universe) of servers as $\mathcal{U}$, $|\mathcal{U}| = n$, i.e. the total number of servers is n.
2. A set system $\mathcal{Q}$ over the $\mathcal{U}$ is a set of subsets of $\mathcal{U}$.
3. A strict quorum system $\mathcal{Q}$ is a set system such that for every $Q_1, Q_2 \in \mathcal{Q}$, $Q_1 \cap Q_2 \neq \emptyset$.
4. A probabilistic quorum system $\mathcal{Q}$ is a set system such that for every $Q_1, Q_2 \in \mathcal{Q}$, there exists a constant ϵ $(0 < \epsilon < 1)$ such that the system has at least probability $1 - \epsilon$ $Q_1 \cap Q_2 \neq \emptyset$.

A probabilistic quorum system is constructed as follows. Given an universe of n servers, a quorum is a random subset of $\gamma\sqrt{n}$ $(\gamma \geq 1)$ servers. Thus,

$$\begin{aligned} \epsilon = P(Q_1 \cap Q_2 = \emptyset) &= (1 - \frac{\gamma\sqrt{n}}{n})^{\gamma\sqrt{n}} \\ &\leq e^{-\frac{\gamma\sqrt{n}}{n}\gamma\sqrt{n}} \\ &= e^{-\gamma^2} \end{aligned}$$

2.4.2 Ferreira et al's Model

Ferreira *et al.* [FRA$^+$05] proposed the use the probabilistic quorum model to describe search in an unstructured P2P network. Given n nodes in the network, an object is replicated $\gamma\sqrt{n}$ times onto a random subset of nodes. A query is also sent to a random subset of $\gamma\sqrt{n}$ nodes. It can then be shown that the probability of finding the desired object of the query is at least $1 - e^{-\gamma^2}$. Clearly as γ increases, the object is replicated over more nodes and the probability of finding the object therefore increases.

2.4.3 Cohen and Shenker's Model

The previous analysis assumed that an object/document is uniformly randomly replicated across nodes in the network. Other replication strategies are also possible. Cohen and Shenker [CS02] provided both a theoretical and empirical analysis of such. Here it is assumed that the n nodes in the P2P network each have capacity ρ, i.e. ρ is the number of files each node can store. Let $R = n\rho$ denote the total capacity of the system. It is further assumed that there are m unique files stored in the P2P system and that each file i is replicated on r_i random nodes. Obviously, $\sum_i r_i = R$, and $mr = R$ if $r_i \equiv r$ is a constant. Let $p_i = \frac{r_i}{R}$ be the fraction of the total system capacity allocated to file i. Finally, let q_i be the normalized query popularity for the ith file. Thus

$$\sum_{i=1}^{m} q_i = 1$$

The search size, z_i, of file i, is defined as the number of nodes searched in response to a query q_i to find file i. Of course, the search size will depend very much on the search strategy used. In [CS02] a random probing model is assumed, i.e. each of many probes randomly selects a node in the network. Thus, each probe has a probability $\frac{r_i}{n}$ of finding the requested file i, and a probability $1 - \frac{r_i}{n}$ of not finding the file. The search size z_i is simply a random variable drawn from a geometric distribution

$$P(z_i) = (1 - \frac{r_i}{n})^{z_i - 1}\frac{r_i}{n}$$

The average search size for file i is

$$\mu_z(i) = \frac{n}{r_i}$$

and the expected search size, μ_z, of all m files is

$$\mu_z = \sum_i q_i \times \mu_z(i) = n\sum_i \frac{q_i}{r_i} = n\sum_i \frac{q_i}{Rp_i} = \frac{n}{R}\sum_i \frac{q_i}{p_i} \tag{2.1}$$

For a uniform replication strategy, $r_i \equiv r$ is a constant and $mr = R$. Thus, the expected search size with uniform replication, μ_z^u, is

$$\mu_z^u = n\sum_i \frac{q_i}{r} = \frac{n}{r}\sum_i q_i = \frac{m}{\rho} \tag{2.2}$$

Note, however, that while this is the expected number of nodes we need to search in order to find an item, Cohen and Shenker do not propose a search strategy that limits the search to this number.

Two alternatives to a uniform replication strategy are also considered. A proportional replication strategy replicates content based on its popularity, i.e. proportional to the number of queries requesting it. Thus popular documents will be found on many nodes while infrequently requested documents will be found on fewer nodes.

Perhaps surprisingly, such a replication strategy results in the same expected search size as the uniform replication. For proportional replication strategy, $r_i = Rq_i$ and the expected search size μ_z^p is

$$\mu_z^p = n\sum_i \frac{q_i}{Rq_i} = \frac{nm}{R} = \frac{m}{\rho}$$

In effect, while popular documents will be found by querying fewer nodes than for a uniform replication strategy, this is balanced by the need to visit far more nodes in order to find unpopular documents.

To minimize the expected search size, we would like to minimize $\sum_i \frac{q_i}{p_i}$ in Equation (2.1). Solving this optimization problem [CS02], we have

$$\frac{p_i}{p_m} = \frac{r_i}{r_m} = \frac{\sqrt{q_i}}{\sqrt{q_m}}$$

Thus,

$$r_i = \lambda\sqrt{q_i} = \frac{R}{\sum_i \sqrt{q_i}}\sqrt{q_i}$$

This is the square root replication strategy which produces the optimal expected search size given by

$$\mu_z^s \quad = \quad n\sum_i \frac{q_i}{\lambda\sqrt{q_i}} = \frac{1}{\rho}(\sum_i \sqrt{q_i})^2 \tag{2.3}$$

Clearly, when the query distribution is uniform, i.e. $q_i = \frac{1}{m}$, Equation (2.3) reduces to Equation (2.2). Typically, the query distribution follows a power law [BYGJ+07], i.e. $q_i = \frac{1}{c}i^{-\varpi}$ where c is the normalization constant. In this case, Equation (2.3) becomes

$$\mu_z^s = \frac{1}{\rho c}(\sum_i \frac{1}{i^{\varpi/2}})^2 \tag{2.4}$$

Analysis of publicly available logs from AOL, indicate that the value of ϖ ranges from 0.8 to 1.0.

2.4.4 BubbleStorm

BubbleStorm [TKLB07a] assumes an unstructured network with size n. An object is replicated to r random a query for that object is sent to z random nodes. Thus, the probability of the query meeting at least one node hosting the desired object is $1-(1-\frac{z}{n})^r \geq 1-e^{-\frac{rz}{n}}$. The bubblecast algorithm used to forward queries is similar to the multiple random walker algorithm proposed in [LCC+02], attempting to reduce network traffic and query latency.

2.5 Probably Approximately Correct Search

The previous work on randomized search looked at the expected search length necessary to find a specific document. Assuming a query is sent to a constant number of nodes, z, we can also ask what the probability of finding a document is. This, and related questions, are addressed in recent papers on probably approximately correct (PAC) search [CFH09, CZFH10]. The PAC architecture considers both an acquisition and a search stage.

During the acquisition stage, it is assumed that there exists a collection, $\mathcal{M}$, of m documents, e.g. the Web, which we wish to index. Given n machines in the network, each machine independently samples ρ unique documents from a collection. These ρ documents on a single machine are referred to as the *local collection*. Note the local collections are *non-disjoint* due to independent sampling of each machine. The union of all local collections in a PAC system is referred to as a *collection sample*.

During the search stage, a query is sent to z machines, and the results returned by the different machines are consolidated and then displayed to the user. If we are searching for a single, specific document d_i, then the probability of retrieving this document is given by

$$P(d_i) = 1-(1-\frac{\rho}{m})^z \tag{2.5}$$

In information retrieval, it is more common to be interested in the top-k retrieved documents. In this case, the correctness of a PAC search is measured by *retrieval accuracy*. If $\mathcal{D}$ denotes the set of top-k documents retrieved when searching the full index, i.e. an exhaustive search, and $\mathcal{D}'$ the set of top-k documents retrieved when querying z nodes, then the retrieval accuracy, a, is defined as

$$a = \frac{|\mathcal{D} \cap \mathcal{D}'|}{|\mathcal{D}|} = \frac{k'}{k}$$

where k' denotes the size of the overlap of the two sets, i.e. $|\mathcal{D} \cap \mathcal{D}'|$.

Clearly, this definition of accuracy is independent of the underlying retrieval model, unlike traditional information retrieval metrics such as precision and recall. It should be clear that accuracy is not a measure of the quality of the retrieved documents, as defined by relevance, or non-relevance. Rather, accuracy is purely a measure of the overlap in the results sets achieved when searching the full index, and a random subset of the index. Thus, the definition of accuracy is valid irrespective of which retrieval model is implemented on the P2P network.

The size of the overlap in the result sets, k' is a random variable drawn from a binomial distribution, and is given by

$$P(k') = (\begin{matrix} k \\ k' \end{matrix}) P(d_i)^{k'} (1 - P(d_i))^{k-k'} \tag{2.6}$$

Since Equation (2.6) is a binomial distribution, the expected value of k' is $E(k') = kP(d_i)$ and the expected retrieval accuracy μ_e is

$$\mu_e = \frac{\mu_{k'}}{k} = \frac{k \times P(d_i)}{k} = 1 - \left(1 - \frac{\rho}{m}\right)^z, \tag{2.7}$$

If we assume that a document is, on average, replicated r times onto different nodes in the network, then the total storage of the network, R, satisfies $R = m \times r$. And the Equation (2.7) can be transformed into

$$\mu_e = 1 - \left(1 - \frac{\rho}{m}\right)^z = 1 - \left(1 - \frac{r}{n}\right)^z \tag{2.8}$$

Equation (2.8) approximates to $1 - e^{rz/n}$ [CFH09]. The product of the replication rate, $\frac{r}{n}$, and the number of nodes queried, z, is the *sample index*, which is the ratio of the number of documents in a sample of z nodes to those in the global collection. We can utilize different combinations of the replication rate, $\frac{r}{n}$, and nodes queried, z, to arrive at the same accuracy. Figure 2.3 illustrates the relationship between accuracy and the sample index. For an expected accuracy of about 80%, i.e. when conducting a search we expect to retrieve, on average, 80% of the documents that would have been retrieved through an exhaustive search, the coverage needs to be about 2. That is, the product, $z \times \ r/n$, must be about twice the number of unique documents in the collection. This can be achieved with very many different combinations of z and $\frac{r}{n}$, which are under the control of the designer. For example, in a network where each node indexes 2% of the global collection, querying 100 randomly selected nodes would provide for the same accuracy as querying 10 randomly selected nodes in a network in which each node indexes 20% of the global collection.

Note that $\frac{\rho}{m} \equiv \frac{r}{n}$ holds only for *uniform* replication. PAC search can also be implemented for proportional and square-root replication [AFC11], and in these cases, the accuracy can be substantially improved.

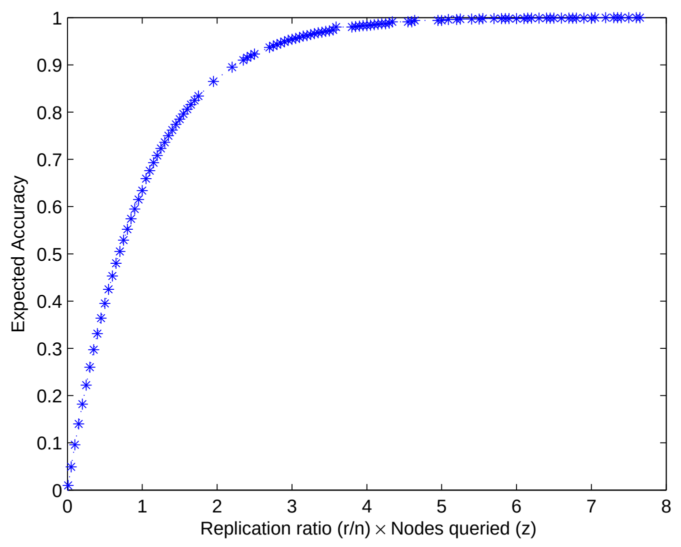


Figure 2.3: The expected accuracy as a function of the product of the number of nodes queried z and replication rate r/n, i.e. *the sample index*.

2.6 P2P Social Networks

The distributed social network, Diaspora[1] addresses the issue of ownership of user data. Users publish data on servers known as *pods* which form the network. Users have complete control over their pod, and can decide how much of their data they wish to share with others. However, if the pod is forced off-line, the data is no longer available.

In the last few years, DHT-based P2P social networks systems such as PeerSoN [BSVD09], eXO [LNTM11], Cuckoo [XCFH10], and My3 [NPA11] have been proposed to address issues of privacy and data availability. However, since the focus of these system is privacy, they do not address how a keyword search across all published data could be performed. Secondly, these systems assume a tight-knit community of peers. If a high popularity user, e.g., a journalist with many followers, were to join such a network, his or her node would become overwhelmed with requests.

[1]https://joindiaspora.com/

Chapter 3

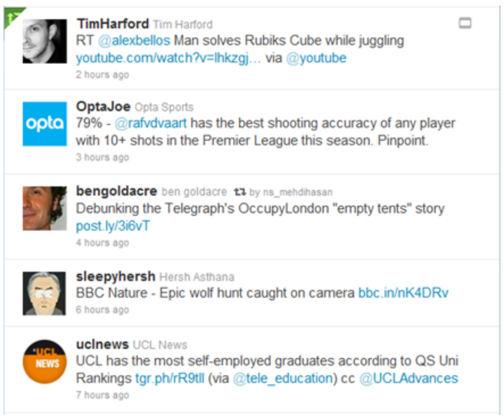
The PAC'nPost Framework

Micro-blogging services, such as Twitter[1], allow users to post short messages for their followers, to retrieve micro-blog posts from users they follow, and to search across publicly posted micro-blog posts (Figure 3.1). We view *following* and *searching* as two instances of information retrieval in which the query is either a user ID or a keyword. Publishing has commonly been seen as independent from retrieval. However, we take a different perspective in which the purpose of publishing is to facilitate retrieval. Using the PAC search architecture, the search of an unstructured network is based on a probabilistic search that queries a fixed number of nodes. Retrieval and search in a dynamic environment such as a decentralized micro-blogging network is then determined by how quickly new information can be randomly distributed through the network. To achieve this we use a modified rumour spreading algorithm.

The probability of retrieving a document is a function of (i) the number of nodes queried, and (ii) the replication of the document, i.e. the number of nodes on which the document is stored. As we increase the number of nodes queried, the latency also increases, so it is desirable to query fewer nodes. For a required probability of retrieval, and a fixed number of nodes to be queried, i.e. a fixed latency, we can analytically derive the minimum number of nodes a document must be stored on using the PAC search architecture. The problem of information retrieval over an unstructured P2P network in which the frequency of information creation is high then becomes how to replicate new information to this number of nodes given the constraints of (i) time (we wish to propagate the information as quickly as possible), and (ii) bandwidth (we wish to propagate the information using as little bandwidth as possible, otherwise our solution will not be scalable). The first constraint could be solved simply by flooding the network with the new information, i.e. sending the information to all nodes in the network. However, such a solution would not satisfy our second constraint. Thus, we seek an algorithm that replicates new information to at least the required minimum number of nodes as quickly as possible, without flooding the network. We emphasize that this replication must occur in the absence of any centralized coordination.

Our framework is one in which published micro-blog posts are replicated onto random nodes in the network. Nodes periodically query a predetermined number of random peers to retrieve the micro-

[1]`https://twitter.com/`


TimHarford Tim Harford
RT @alexbellos Man solves Rubiks Cube while juggling youtube.com/watch?v=Ihkzgj... via @youtube
2 hours ago

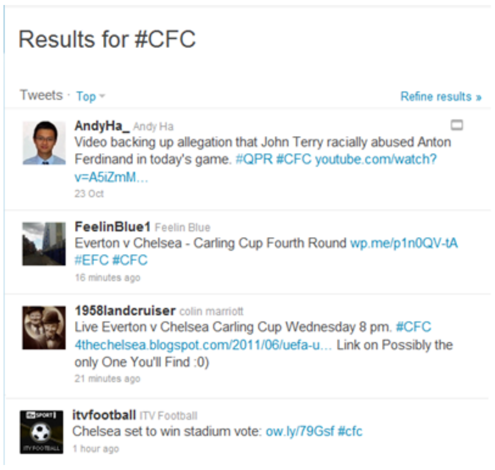
OptaJoe Opta Sports
79% - @rafvdvaart has the best shooting accuracy of any player with 10+ shots in the Premier League this season. Pinpoint.
3 hours ago

bengoldacre ben goldacre by ns_mehdihasan
Debunking the Telegraph's OccupyLondon "empty tents" story post.ly/3i6vT
4 hours ago

sleepyhersh Hersh Asthana
BBC Nature - Epic wolf hunt caught on camera bbc.in/nK4DRv
6 hours ago

uclnews UCL News
UCL has the most self-employed graduates according to QS Uni Rankings tgr.ph/rR9tll (via @tele_education) cc @UCLAdvances
7 hours ago

A Twitter timeline showing Tweets of followed users in reverse chronological order.


Results for #CFC

Tweets · Top
Refine results »

AndyHa_ Andy Ha
Video backing up allegation that John Terry racially abused Anton Ferdinand in today's game. #QPR #CFC youtube.com/watch?v=A5iZmM...
23 Oct

FeelinBlue1 Feelin Blue
Everton v Chelsea - Carling Cup Fourth Round wp.me/p1n0QV-tA #EFC #CFC
16 minutes ago

1958landcruiser colin marriott
Live Everton v Chelsea Carling Cup Wednesday 8 pm. #CFC 4thechelsea.blogspot.com/2011/06/uefa-u... Link on Possibly the only One You'll Find :0)
21 minutes ago

itvfootball ITV Football
Chelsea set to win stadium vote: ow.ly/79Gsf #cfc
1 hour ago

Search Results for query *#CFC*

Figure 3.1: Timeline and search on Twitter

blog posts they are following. This mechanism is in direct contrast with a Publish-Subscribe (pub/sub) architecture. In a pub/sub architecture, publishers can *push* any new data to their subscribers. This has some advantages over a framework like PAC'nPost. In a pub/sub system, the published data need not be replicated, and can be instantaneously be delivered to the subscribers, who do not have to periodically query to check for new micro-blog posts. Xu *et al.* [XCFH10] describe one such system. However, as we show in Section 3.3, PAC'nPost can retrieve a published micro-blog post within a minute of it's publication. Furthermore, implementing a micro-blogging social network in a P2P network via a pub/sub system has the following disadvantages:

- Publishers will need to maintain an up-to-date list of all the nodes who follow them. In a micro-blogging social network, users follow and unfollow other peers regularly. This would place an unfair burden on popular publishers. If the user-follower matrix was sub-contracted to other nodes, these management nodes would become a point of vulnerability for the system as attacking these nodes would incapacitate the system.

- When a publisher node goes offline, a newly joined subscriber node will not be able to retrieve the publisher's micro-blog posts unless a caching mechanism is implemented.

- The pub/sub system provides no mechanism for keyword search across all published micro-blog posts. A node which wishes to perform a keyword search would have to search all the other nodes in the network, which is, of course, impractical.

The rest of this chapter is organized as follows: In Section 3.1 we provide the overview of our framework. Section 3.2 then outlines the assumptions required to implement our framework. In Section 3.3 we outline the behaviour of nodes in the network and provide summaries of simulations that support our design. In Section 3.4 we investigate an alternate strategy, one which can withstand spikes in the data creation rate. Section 3.5 provides estimates for the bandwidth requirements on the nodes participating in the network. Finally 3.6 summarizes our results and discusses future work.

3.1 Framework Overview

Our goals are two-fold. First, a user should be able to retrieve the micro-blog posts[2] of other users he/she is *following*, with a sufficiently high accuracy by querying z random nodes in the network. Second, a user should be able to perform a keyword search, akin to a search engine, by sending the query to, again, z random nodes. To retrieve the required posts, a node in the network makes a *request* to z other nodes every s seconds. By randomizing the request time at each node, we can ensure that the number of nodes making a *request*, at any given moment, is roughly equal.

The key features of PAC'nPost are:

- PAC'nPost is a framework which describes an overlay on top of an unstructured P2P network, and does not take into account the network topology in its operations.
- Nodes periodically query a set number of random peers to retrieve the micro-blogs they follow. In each query, nodes specify the peers they *follow*. Nodes are unconcerned about their *followers*, i.e., who is following them.
- Nodes query a relatively small number of peers. To facilitate the querying of nodes, PAC'nPost requires an underlying membership and peer sampling service which provides each node with a partial view where other nodes are uniformly sampled from the network, and each node's partial view is frequently refreshed by partially or fully swapping with another node's view. Cyclon [VGVS05], the overlay management system proposed by Jelasity *et al.* [JVG+07], and Brahms [BGK+09] are examples of such services.

To participate in the network, each node contributes some disk space. Of this disk space, a proportion is utilized for storage of the posts, and the remaining proportion for indexing the stored posts.

The load on individual nodes can also be calculated as follows: Assuming that roughly the same number of peers $\left(\frac{n}{s}\right)$ issue a *request* at any given second, and each peer selects z random nodes for its request, the probability of being picked is simply $(\frac{z}{n})$, and therefore the number of requests any random node in the network is expected to receive per second is $\frac{n}{s} \times \frac{z}{n} = \frac{z}{s}$.

Previous work on gossip protocols, has focused on one rumour - propagating spreading software updates in a network of servers or tracking the progress of a marketing campaign across online social networks [DGH+87, LAH07, NMBM07]. These rumours are not impeded by rival concurrent rumours. In our framework, multiple blog posts are created every second, each of whom need to be replicated onto the required number of nodes. Thus, our framework needs to facilitate *multiple simultaneous rumours*. Furthermore, rumour spreading usually has the goal of ensuring that the rumour spreads to the entire network at the fastest possible rate. Our requirement is subtly different; we require that a document be replicated rapidly only onto a fraction of the network. We call this *rapid-yet-restrained* dissemination.

[2]Henceforth, *blog* is used interchangeably with *micro-blog* and *post(s)* is used as an abbreviation for *micro-blog post(s)*

Our framework utilizes the PAC search architecture for retrieval, and we briefly recap the key parameters as follows:

- n - the number of nodes in the network
- m - the number of unique documents
- ρ - the capacity of each node, i.e. the number of unique documents it can store
- r - the number of nodes a document is replicated onto
- z - the number of nodes queried to retrieve a document

The correctness of a PAC search is measured by the *accuracy*, μ and is equal to

$$\mu = 1 - \left(1 - \frac{\rho}{m}\right)^z = 1 - \left(1 - \frac{r}{n}\right)^z \tag{3.1}$$

The ratio, $\frac{r}{n}$ is referred to as the replication rate. Note that $\frac{\rho}{m} \equiv \frac{r}{n}$ holds only for *uniform* replication.

The accuracy applies to both blog post retrieval, which is a case of *known item* search as well as to keyword search, which performs a word/phrase search on the entire document collection. Since accuracy is key to our framework, we provide a brief discussion on its use as follows: Let us assume that the accuracy of the system is 90%, and we examine any random node in the network, say, Node $\mathcal{A}$.

In the case of blog post retrieval, we know in advance which documents we need to retrieve (i.e. the ones which are published by the peers followed by Node $\mathcal{A}$). Let us also assume that Node $\mathcal{A}$ follows 10 other nodes in the network, all of whom have published a blog post. When Node $\mathcal{A}$ makes its first request upon joining the network to z other random nodes, the accuracy would suggest that Node $\mathcal{A}$ would have retrieved 9 out of the 10 blog posts it follows, missing one. In a system, each node makes a request every s seconds, and when Node $\mathcal{A}$ makes its second request, it would specify in the request, the identifier of the latest blog post of each followed peer that it has successfully retrieved from its previous request, thereby requesting the missing blog post and any new blog posts published by the peers it follows. The chances that it misses the same blog post are low, and therefore, with high probability, specifically 99%, it would have retrieved all the 10 blog posts it follows by the second request. The second request, is a scheduled request, as nodes have to query periodically for new blog posts in a micro-blogging social network.

Now let us consider the case of keyword search. When Node $\mathcal{A}$ makes a keyword search by querying z other nodes, each of the responders would contain the top-10 (or top-k, where k is a system parameter) blog posts ranked in descending order of relevance score. The results of the z nodes would be merged into a final top-10, which would be displayed to the user at Node $\mathcal{A}$. An accuracy of 90% implies that this top-10 would contain 9 of the 10 blog posts out of a result if Node $\mathcal{A}$ had queried the entire network. Note that in the case of keyword search, accuracy does not consider the rank order of the retrieved blog posts.

3.2 Assumptions

In this section we detail the assumptions required for our framework. We make two major assumptions, which are as follows:

- We assume that the required accuracy, as defined earlier, is 95%. An accuracy of 95% implies that on average we will retrieve 95% of the posts we want by querying z nodes, as compared to searching the entire network. Choosing the required accuracy is the most important decision that the designer of a PAC'nPost system has to make. For our experiments in this chapter as well as all the PAC'nPost extensions described in the successive chapters, we set the accuracy at 95%.

  The designer of a PAC'nPost system may choose to aim for a lower accuracy in order to reduce the amount of replication or the number of nodes queried, but we chose a value high enough to demonstrate that our framework can facilitate P2P micro-blogging with high levels of accuracy. An accuracy of 95% seems sufficiently high that most users will be satisfied with the performance. Note that the accuracy is for one request, which consists of querying z randomly selected nodes. *Performing another request for the same information increases the accuracy further.*

- The number of nodes queried is fixed at 25. The decision to query 25 nodes reflects a compromise. Querying a larger number of nodes will increase the probability of finding a post, but at the expense of network bandwidth and latency. Although we have not directly considered latency in our analysis, we believe that querying 25 nodes does not result in significant latency.

  As we discuss briefly, at the end of this section, we can envisage a PAC'nPost system with different values for the number of nodes queried, which arrives at the same accuracy. In Chapter 4, we derive the number of nodes queried in order to minimize the overall system bandwidth. However, the derived number of nodes queried come at the expense of a more complex system with a larger number of parameters.

We can rearrange Equation (3.1) to extract the replication ratio:

$$R = \frac{r}{n} = 1 - \exp\left(\frac{\log(1-\mu)}{z}\right) \tag{3.2}$$

Setting the accuracy to 95%, i.e., $\mu = 0.95$ and the number of nodes queried to 25, i.e., $z = 25$, we get $\frac{r}{n} = 0.12$, i.e. a post needs to be replicated to 12% of the network.

Since nodes query a relatively small number of peers, we assume message latencies of a single time unit. We assume each iteration is equivalent of one second and we use the two interchangeably. An iteration is the smallest unit of PAC'nPost's observational window.

We assume that the interval between requests, s, is 30 seconds, which allows for a user experience that is similar to Twitter. Since each node makes a request every $s = 30$ seconds, 95% accuracy does not imply that a user will miss 1 in 20 of his/her followed posts. The accuracy is indeed 95% at the first request, but increases to 99.83% by the second request, and 99.99% by the third request, since making two requests to two sets of z random nodes is the same as making one request to $2z$ random nodes as long as the same information is requested.

When a post is published, it must be given some time to spread through the network before it can be retrieved. We allow every post $s_r = 30$ seconds to replicate onto nodes and be available for retrieval. Note, this is somewhat arbitrary.

Next, we define a micro-blog post as text limited to 500 characters including white-space. Note, this is approximately 3.5 times larger than a Twitter message and is large enough for a small paragraph of text or a couple of sentences with a URL. We assume UTF-8 [3] encoding of blog posts, and on average, 2 bytes per character, and thus each post requires 1 KB in disk space.

We assume that each node in the network contributes, on average, 1GB of disk space to support the services. Of this contributed disk space, 90% is utilized for storage of the posts, and the remaining 10% for indexing the stored posts. In the storage area, once the capacity has been exhausted, posts are processed in a First In First Out manner, i.e., older posts are replaced by younger ones. The node stores newer micro-blog posts in uncompressed form as they are more likely to be retrieved when it is queried by other peers performing their scheduled queries, and compresses older posts in batches in order to store the maximum number of posts.

The allocation of 900 MB to the storage of posts allows 9 million posts to be stored on each node, assuming a compressed micro-blog post requires 0.1 KB[4]. Since the replication rate is 12%, the total number of *unique* posts that can be stored is 75 million[5]. Note that this number can be increased by reducing the replication rate and correspondingly increasing the number of nodes queried, i.e. there is a direct trade off between communication and storage. Assuming a network of 100,000 users and an average post creation rate of 2.5 posts per second[6], the system would have the capacity to store all the blog posts created during a period of 340 days, which is probably an acceptable storage capability.

For the selected values of our parameters, on average any random node in the network would have to answer $\frac{z}{s} = 0.83$ requests per second or 50 requests per minute. Note, this threshold is for one set of parameter values. We could envisage a network where the required accuracy is 95%, with $s = 60$, and $z = 50$. In this case, the load on the nodes would be the same, but the posts would only need to be spread to 6% of the network. However, in this case, the posts would only be retrieved once per minute.

3.3 Node Behaviour

In a micro-blogging social network, many blog posts are created at any given moment, and we describe how this is managed by the system. When a node queries other nodes, or is queried by another node, it transfers its most recent post, thereby spreading the post into the network. Unfortunately, sending only the most recent post to contacted nodes does not spread the post into the network at an acceptable rate. To increase the rate of spreading, we introduce a *transfer buffer*, $\mathcal{T}$, which stores a small fraction of the most recent blog posts that the peer has encountered. The transfer buffer has a fixed size, $\mathcal{T}_L$.

[3] `http://tools.ietf.org/html/rfc3629`

[4] Clearly, an individual post cannot be compressed to 10% of it original size, but as multiple posts are stored in the storage area, 90% compression can be easily accomplished.

[5] Since, as previously stated, the replication rate ($\frac{r}{n} = \frac{\rho}{m}$), where ρ is the number of documents stored on a node and m is the total number of unique documents in the network. Thus, $m = 75$ million.

[6] Section 3.3.2 contains the discussion on blog post creation rates.

To retrieve the posts Node $\mathcal{A}$ is *following*, it makes a *request* to z other nodes every s seconds. A request consists of

- the list, $\mathcal{L}_A$, of the user IDs Node $\mathcal{A}$ follows
- the most recent post created at Node $\mathcal{A}$
- the posts contained in the *transfer buffer* $\mathcal{T}_A$ of Node $\mathcal{A}$

When Node $\mathcal{B}$ receives a *request*, it sends back a *response*, which consists of

- the most recent posts of all the peers specified in $\mathcal{L}_A$, which are found in its storage area
- the most recent post created at Node $\mathcal{B}$
- the posts contained in its *transfer buffer* $\mathcal{T}_B$

In addition to the *request* and *response*, when a blog post is created at a node, it immediately performs a *request* to z nodes. By performing a *request* at the time of post creation, the newly created post immediately starts to spread through the network.

When a node, say Node $\mathcal{A}$, makes its scheduled request, it always specifies the peers it is following in list $\mathcal{L}_A$. Thus, if it has unfollowed some peers or has followed new peers, the correct list is always included in the request. The queried nodes search their storage areas only for the micro-blog posts of the peers that Node $\mathcal{A}$ is following.

In our setup, the scheduled queries retrieve the followed posts, but are also used to replicate posts via the transfer buffer. There exists an underlying assumption that each node follows at least one other peer. This is not unreasonable as even users with a large number of followers on Twitter (e.g. Barack Obama, BBC Breaking News) follow a small number of other users. However, we can consider a situation where Node $\mathcal{A}$ publishes a blog post and contacts z random nodes. If each of these z random nodes do not follow any other peers, they will not be performing scheduled queries to retrieve followed posts. In this very unlikely scenario, the blog post will fail to replicate in the network.

At the completion of a *request*, the responders store the contents of the request, i.e. the contents of the requester's transfer buffer and the requester's most recent post in their storage area. Similarly, the requester stores the contents of the responders' transfer buffers and the responders' most recent posts in its storage area, as well as the posts which the node follows and which were returned by the responders.

The storage area consists of posts stored in ordered lists such that each list contains posts created at the same time and the lists are sorted in descending time order. Whenever a new post is added to a storage list, the contents of the list are randomized. The purpose of the randomization is to support the construction of the transfer buffer, discussed next.

After responding to a request or processing all the responses from the z queried nodes, each node reconstructs its transfer buffer.

3.3.1 Transfer Buffer

The goal of selecting posts for the transfer buffer is to continue spreading newer posts at the expense of older posts. In the terminology of gossip protocols, blog posts which are not selected into a node's buffer are *stifled*, i.e. the node has received the post but refuses to disseminate it further in the network.

A number of strategies for selecting the posts for the buffer are possible. We investigated two, namely most recent, and a probabilistic strategy in which the probability of selection is proportional to the age of the post.

Since each post is allowed s_r iterations to spread in the network, we consider only the top-s_r lists in the storage area when constructing the transfer buffer. We investigated deterministic and probabilistic approaches to select posts such that $\mathcal{T}$ is filled to capacity. In the deterministic method *(DSel)*, posts are read off the sorted lists in chronological order until $|\mathcal{T}| = \mathcal{T}_L$. The probabilistic method attempts to give older posts a chance to be selected in the transfer buffer by assigning probabilities to each $\mathbf{L}_j$ such that

$$P(\mathbf{L}_j) = \frac{i^{-j}}{c}$$

where c is a normalization constant. We investigated probabilistic selection for $i = 1.5$ *(PSel1.5)* and $i = 2.0$ *(PSel2.0)*. When $i = 2.0$, the most recent blog post has twice the chance of being selected than a post created one iteration earlier.

3.3.2 Simulations

To verify our theoretical framework, we performed simulations based upon the parameter values discussed in the previous section. As described in Section 3.1, our framework is an overlay on top of an unstructured P2P network, and nodes can contact randomly chosen peers via this overlay without consulting the underlying network topology.

Nodes query $z = 25$ other random peers every $s = 30$ iterations. At each iteration, 25 new posts are created at random nodes in the network. This is the normalized rate of post creation for a network of 1M nodes based upon Twitter's average of 4,600 tweets per second with a 200 million user base [Tec12]. We also conducted simulations where 125 posts are created per second. The simulations are based on network sizes of 10,000, and 100,000 nodes. A post creation rate of 25 posts per second on a network of 10,000 and 100,000 nodes corresponds to 100 and 10 times Twitter's average respectively. Similarly a post creation rate of 125 posts per second on a network of 10,000 and 100,000, nodes corresponds to 500 and 50 times Twitter's average. We chose post creation rates magnitudes higher than that of Twitter to demonstrate that our framework can facilitate high data rates without overwhelming any given node. Running the system with lower data creation rates is as with the higher data creation rate, but with lower requirements on nodes and lower overall system bandwidth.

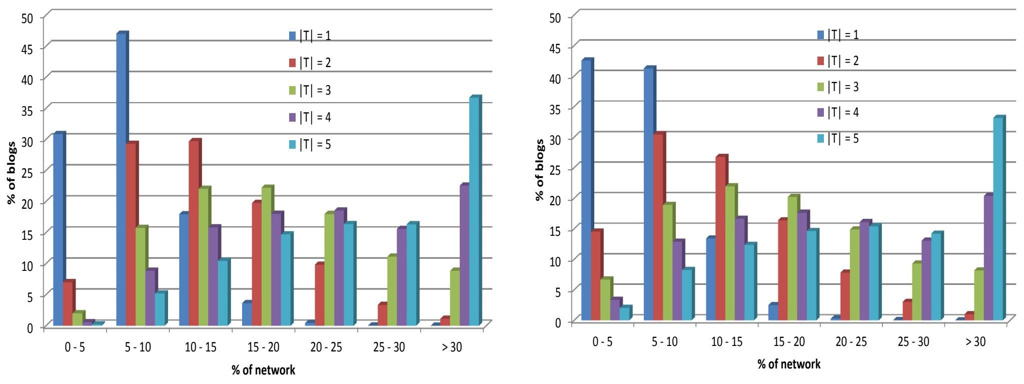
We used the TREC Micro-blog Tweets2011 corpus[7] [OMLS11] as the source of posts for each node, i.e. when a node created a post, it sampled, without replacement, a random post from the collection.

To evaluate the effect of the transfer buffer size and three methods for constructing the buffer, namely deterministic (*DSel*), and probabilistic with $i = 1.5$ (*PSel1.5*) and $i = 2.0$ (*PSel2.0*), we per-

[7] http://trec.nist.gov/data/tweets/

Table 3.1: Information retrieval and dissemination with a post creation rate of 25 posts per second for transfer buffer size $|\mathcal{T}| = 3$ and a network of size $n = 100,000$

Selection	Accuracy		Dissemination	
Method	Mean	Std Dev	Mean	Std Dev
DSel	96.676	0.147	16.635	8.699
PSel2.0	96.565	0.177	16.243	8.956
PSel1.5	96.487	0.149	16.364	9.200



(a) $n = 10,000$ (b) $n = 100,000$

Figure 3.2: The dissemination of posts with a post creation rate of 25 posts per second for networks of 10,000 and 100,000 nodes.

formed 10 simulations for each value of network size (10,000 and 100,000 nodes), various transfer buffer sizes, and transfer buffer selection methods (*DSel, PSel1.5, PSel2.0*). Each simulation lasted for 600 iterations, so that each node in the network would make at-least 20 *requests*. Under steady-state conditions, i.e. when all transfer buffers are full, we recorded each post's replication rate at the end of each simulation.

Retrieval accuracy was also measured. We assumed each node follows 35 randomly chosen user IDs. Though the total number of follower-links in the Twitter graph is not known, Kwak *et al.* [KLPM10] reported 1.47 billion links for 41.7 million users. Note that we only consider posts that have been in existence for at least $s_r = 30$ seconds in order to allow for dissemination across the network.

We found that the results of *DSel*, *PSel1.5*, and *PSel2.0* were almost identical, with the deterministic selection method performing marginally better. Table 3.1 displays the summary of simulations performed on a network of 100,000 nodes and a transfer buffer of size 3 with a post creation rate of 25 posts per second.

The results of all the simulations are summarized in Tables 3.2 and 3.3. The mean and standard deviations are the average over the 10 trials, for each configuration. The figures presented in the tables are for deterministic selection (*DSel*); for maintaining clarity we omit the probabilistic selection methods

Table 3.2: Information retrieval and dissemination with a post creation rate of 25 posts per second in networks of 10,000 and 100,000 nodes

$n = 10,000$				
$\|\mathcal{T}\|$	Retrieval Accuracy		Dissemination (% of network)	
	Mean	Std Dev	Mean	Std Dev
1	90.652	0.421	7.371	3.752
2	**97.045**	**0.177**	**13.055**	**6.236**
3	98.701	0.114	18.058	8.182
4	99.264	0.100	22.514	9.736
5	99.486	0.088	26.564	10.945

$n = 100,000$				
$\|\mathcal{T}\|$	Retrieval Accuracy		Dissemination (% of network)	
	Mean	Std Dev	Mean	Std Dev
1	83.485	0.439	6.331	3.724
2	93.509	0.180	11.807	6.518
3	**96.676**	**0.147**	**16.635**	**8.699**
4	98.074	0.130	20.947	10.468
5	98.775	0.090	24.925	11.694

Table 3.3: Information retrieval and dissemination with a post creation rate of 125 posts per second in networks of 10,000 and 100,000 nodes

$n = 10,000$				
$\lvert\mathcal{T}\rvert$	Retrieval Accuracy		Dissemination (% of network)	
	Mean	Std Dev	Mean	Std Dev
4	93.383	0.104	7.498	3.175
5	**95.270**	**0.064**	**9.014**	**3.842**
6	96.527	0.118	10.478	4.411
7	97.348	0.061	11.904	4.935
8	97.846	0.033	13.281	5.498

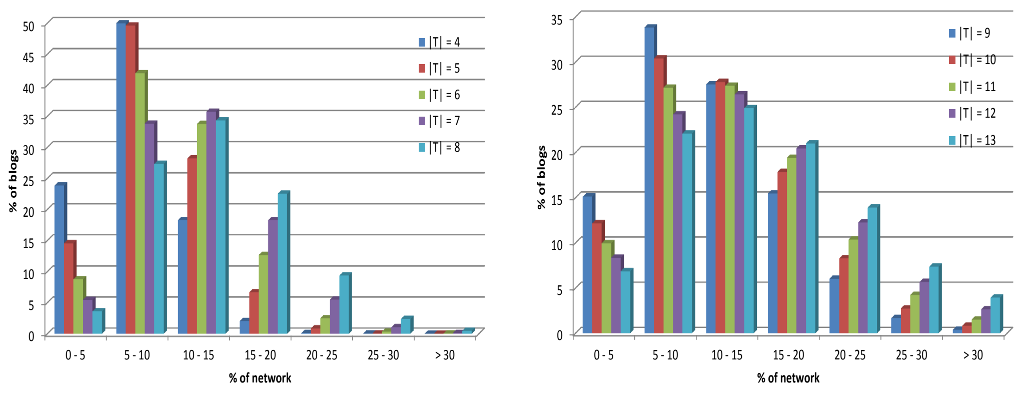
$n = 100,000$				
$\lvert\mathcal{T}\rvert$	Retrieval Accuracy		Dissemination (% of network)	
	Mean	Std Dev	Mean	Std Dev
9	93.583	0.121	11.025	5.833
10	94.676	0.092	12.075	6.283
11	**95.468**	**0.061**	**13.118**	**6.732**
12	96.082	0.050	14.128	7.185
13	96.596	0.121	15.119	7.601

Table 3.4: Iterations required for posts to reach 95% of their maximum replication for a post creation rate of 25 posts per second in networks of 10,000 and 100,000 nodes

	$n = 10,000$					$n = 100,000$				
$\lvert\mathcal{T}\rvert$	1	2	3	4	5	1	2	3	4	5
Iteration	24	18	14	14	13	15	16	16	17	17

Table 3.5: Iterations required for posts to reach 95% of their maximum replication for a post creation rate of 125 posts per second in networks of 10,000 and 100,000 nodes

	$n = 10,000$					$n = 100,000$				
$\lvert\mathcal{T}\rvert$	4	5	6	7	8	9	10	11	12	13
Iteration	23	22	19	16	14	15	15	15	15	15



(a) $n = 10,000$ (b) $n = 100,000$

Figure 3.3: The dissemination of posts with a post creation rate of 125 posts per second for networks of 10,000 and 100,000 nodes.

(*PSel1.5, PSel2.0*), the results of which are very similar. Figures 3.2 and 3.3 provide the breakdown of the post dissemination values in Tables 3.2 and 3.3 respectively. The aforementioned tables and figures show that as the size of the transfer buffer increases, the percentage of posts which spread to a higher percentage of the network increases, as expected. For a post creation rate of 25 posts per second, a transfer buffer size of 2 for network of 10,000 nodes, or a buffer size of 3 for a network of 100,000 nodes, would satisfy our retrieval requirement, i.e. an accuracy of 95%. Notice that while the size of the network increased by 10 times, we only needed to increase the size of the transfer buffer by one post. When the post creation rate increases to 125 posts per second, we require transfer buffer sizes of 5 and 11 for networks of 10,000 and 100,000 nodes respectively.

Figures 3.2 and 3.3 illuminate the percentage of posts that are copied to 0-5%, 5-10% etc. of the network. We observe that, for example, in Figure 3.2, for a network size of 10,000 and a transfer buffer size of 2, the number of posts that are copied to greater than 30% of the network is just over 1% of posts. Similarly for a transfer buffer size of 3, and network size of 100,000 nodes, the percentage of posts propagating to greater than 30% of the network is just over 8%. This demonstrates that the rumour propagation algorithm exhibits constrained dissemination, i.e. posts are copied to a sufficient number of nodes to guarantee a required accuracy of 95%, but the number of posts copied to fewer of more nodes is small.

The nature of blog post selection into the buffer facilitates the blog post to spread rapidly in the network when it is young, achieving 95% of its final dissemination in the first 14-16 iterations, after which it replicates much slower due to the low probability of being selected into the transfer buffer. Tables 3.4 and 3.5 show the number of iterations by which *all* posts reach 95% of their final replication. The tables show that for a small network size (10,000 nodes), the effect of increasing the size of the transfer buffer reduces the iterations required to attain near-maximal value. This is explained by the fact that the posts have more chance of being selected into the transfer buffer as the buffer size increases. However, the relatively small size of the network increases the probability of being spread to a node where the post already exists, and so the post replication converges quickly to its near-maximal value. For a larger network of 100,000 nodes, increasing the size of the transfer buffer increases the probability of the posts being selected into the buffer, but the posts also have more nodes to replicate onto and therefore require an extra iteration to attain near-maximal replication. The simulations show that even in large networks, we can attain rapid post dissemination by selecting an appropriate value of the transfer buffer.

3.4 Flexible Transfer Buffer

In section 3.3 we provided a basic framework for micro-blogging in a peer-to-peer network. From our simulations, it is clear that the required size of the transfer buffer depends on the blog post creation rate. In this section, we investigate a flexible buffer strategy where the buffer can adapt to the data creation rate. We also introduce keyword search in our simulations, and we subject our framework to various types of churn.

To enable the flexible transfer buffer, during the transfer buffer creation process, instead of reading off the most recent posts until the transfer buffer reaches a certain size (*DSel*), posts are selected into the buffer with an exponentially decreasing probability which is a function of the age of the post:

$$P(sel) = e^{-t/\beta}$$

where t is the number of seconds the post has been in existence and β is a system parameter. For instance, when $\beta = 15.7$, a post is which is 1 second old has a probability of almost 94% of being selected into the buffer, but a post which is 20 seconds old only has a the probability of 28% of being selected into the transfer buffer. Note, once a post is not selected into the transfer buffer, it is permanently set for no further selection, even if it's age is less than $s_r = 30$ seconds. Another way of stating this probabilistic selection process, is that at every iteration a node becomes a *stifler* with a probability $1 - e^{-t/\beta}$. Conceptually, β can be thought of as the parameter which sets the probability of a node passing the rumour to another.

The node behaviour, i.e. the scheduled request containing the required retrieval information, transfer buffer, etc. and the response remains the same as in Section 3.3. To perform a keyword search, the list $\mathcal{L}$ is replaced by the keyword query in the *request*, and the *response* contains the posts matching the query, which are then merged and re-ranked at the query originating node.

3.4.1 Simulations

To test the flexible buffer, we set up a network as before. Nodes queried $z = 25$ other random peers every $s = 30$ iterations, each node followed 35 other random peers, and the TREC Micro-blog Tweets2011 corpus was used as the source of posts for each node.

At each iteration, 25 new posts were created at random nodes in the network. We also introduced a spike in the post creation rate where 125 posts are created at random nodes temporarily. This is motivated by the fact that on Twitter, a five-fold increase in tweets-per-second is known to happen when important events occur. For completeness, we also introduced a lull period where the post creation rate drops to 5 posts per second. The simulations are based on network sizes of 10,000 and 100,000 nodes. A creation rate of 25 posts per second on a network of 10,000 and 100,000 and nodes corresponds to 100 and 10 times Twitter's average.

To evaluate the accuracy of keyword search, we generated a selection of keywords from the collection. In order to measure the accuracy of keyword search, each time a node publishes a post, it is also stored in a central database. When a node performs a keyword search, the results obtained by the node are compared to the results obtained searching the centralized database, the latter serving as a gold standard, being equivalent to performing an exhaustive search of all nodes in the network.

A node that received a keyword query performed a search of its local collection using the BM25 ranking algorithm. Experiments are reported for the cases where the parameters of the BM25 algorithm are set based on (i) the statistics of a node's local collection, and (ii) the statistics of the global collection of tweets. The querying node received the ranked lists from each of the z queried nodes and merged them based on the BM25 scores each node provided. The baseline search of the centralized database is performed using BM25 and the statistics of the global collection.

To evaluate the effect of the dissemination parameter, β, we performed 10 simulations for each value of β and network size (10,000 and 100,000 nodes). Each simulation lasted for 600 iterations, so that each node in the network would make at-least 20 *requests*. At each iteration of each simulation, we recorded the number of posts in each node's transfer buffer ($|\mathcal{T}|$). In our simulations, when a node made a *request* to retrieve its required posts, it also included a keyword search within the request. For every node which made a *request*, at an iteration, we recorded the accuracy, as defined in Section 3.1 for both post retrieval and keyword search. The accuracy values reported are averaged over all iterations of all simulations of each dissemination parameter (β) and network size (n) combination. To calculate the dissemination of posts, we recorded each post's replication at the end of each simulation.

Table 3.6 illustrates the accuracy for retrieval of *followed* posts and retrieval based on keyword search. As per system design, we only consider posts that have been in existence for at least $s_r = 30$ seconds in order to allow for dissemination across the network. The keyword accuracy @ k is the proportion of posts in the top-k of the merged results of querying z nodes in comparison with the top-k results of performing the identical search on the centralized database.

We note that the accuracy of keyword search is lower than post retrieval (following) accuracy. This is due to the fact that the BM25 ranking algorithm requires parameters (e.g. the number of documents in the collection, the number of documents matching a query term) that are calculated from each node's local collection. These values differ from the corresponding values of the global collection. To confirm this, we re-ran our simulations and made the global values of the BM25 parameters available to all nodes. Table 3.7 illustrates the keyword search accuracies with global BM25 parameter values. The values are much closer to the expected 95$\%$ accuracy.

For network sizes of 10,000 and 100,000 nodes a dissemination parameter, β, value of 5.1 and 14.8 respectively provides the required 95$\%$ retrieval accuracy. Table 3.8 shows the replication rate of the posts, for the two network sizes. As the value of β increases, the percentage of posts which spread to a higher percentage of the network increases, as expected. We observe that, for example, for a network

Table 3.6: Accuracy as a function of the dissemination parameter, β. Note that the keyword search is performed using BM25 with parameters set using a node's local statistics.

Network size (n) = 10,000								
β	Blog Post Retrieval		Keyword Search					
			@ 10		@ 20		@ 30	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
4.9	94.701	4.386	80.797	4.112	83.632	4.570	85.117	4.739
5.0	95.075	4.309	81.570	4.679	84.078	4.720	85.569	4.102
5.1	**95.327**	**4.512**	81.940	4.648	84.538	4.120	85.965	4.170
5.2	95.797	3.753	82.586	3.968	85.101	3.692	86.469	4.065
5.3	96.145	2.972	83.477	3.269	85.920	3.184	87.021	2.853

Network size (n) = 100,000								
β	Blog Post Retrieval		Keyword Search					
			@ 10		@ 20		@ 30	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
14.6	94.885	3.145	84.414	3.336	86.697	3.419	87.828	3.099
14.7	95.032	2.919	84.296	2.760	86.566	3.195	87.809	2.720
14.8	**95.457**	**3.230**	84.970	3.213	87.277	2.922	88.361	3.509
14.9	95.552	2.919	85.036	2.692	87.381	3.064	88.684	2.915
15.0	95.963	2.921	85.486	2.805	87.767	2.657	89.085	2.888

size of 100,000 nodes and $\beta = 14.8$, that, on average, posts are propagated to just over 12% of the network, which is very close to the theoretically desired fraction of 12%. As with the system employing the fixed-size transfer buffer, the number of posts that are copied to greater than 30% of the network is less than 1% of the posts. This demonstrates that the rumour propagation algorithm exhibits constrained dissemination, i.e. posts are copied to a sufficient number of nodes to guarantee a required accuracy of 95%, but the number of posts copied to fewer of more nodes is relatively small.

Whilst the dissemination parameter, β, controls the selection of posts into the transfer buffer, and consequently the replication of posts in the network, the communication load incurred by the nodes depends on the size of the transfer buffer, $|\mathcal{T}|$. Figures 3.4(a) and (b) illustrate the post creation rate, retrieval accuracy, and the mean and maximum buffer size as recorded at each iteration (averaged over all simulations) for a network of 100,000 nodes and a dissemination parameter value $\beta = 14.8$. The important point to note is that the accuracy remains at around 95% even when there is a spike or lull in the post creation rate, and our proposed system can automatically deal with spikes which are encountered in actual centralized systems such as Twitter.

Figures 3.4(a) and (b) also illustrate that the mean and maximum value of $|\mathcal{T}|$ increases a few

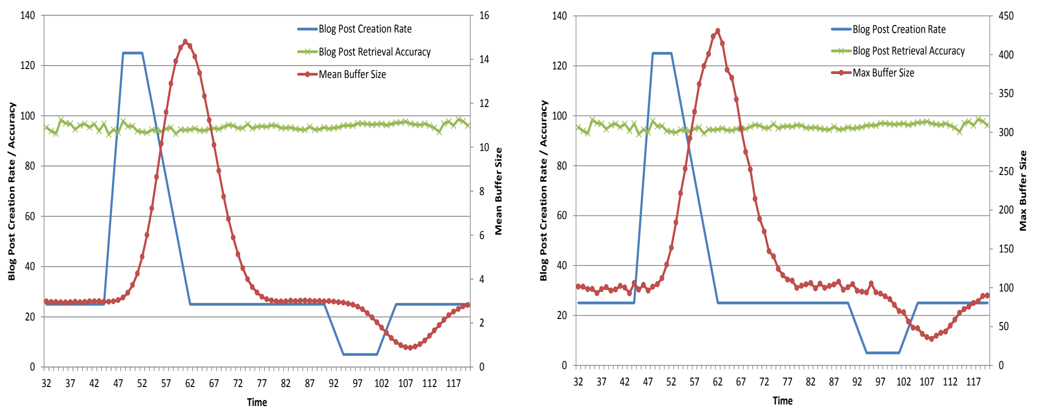
Table 3.7: Accuracy as a function of the size of the dissemination parameter, β, for a network size of 100,000 nodes, when each node has access to global BM25 parameter values

β	Blog Post Retrieval		Keyword Search					
			@ 10		@ 20		@ 30	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
14.7	95.032	2.919	91.990	2.655	92.454	2.695	92.645	2.769
14.8	95.457	3.230	92.796	3.362	93.284	3.274	93.338	3.105
14.9	95.552	2.919	92.838	3.047	93.365	3.003	93.628	2.981

Table 3.8: Replication rate (% of network) as a function of the dissemination parameter, β.

Network size (n) = 10,000									
β	Overall		Fraction of posts replicated to between x-y% of the network						
	Mean	Std Dev	0-5	5-10	10-15	15-20	20-25	25-30	>30
4.9	10.651	5.913	14.488	40.620	24.859	11.595	5.562	2.107	0.769
5.0	10.966	6.074	13.703	38.936	25.787	12.439	5.889	2.288	0.958
5.1	11.615	6.509	12.656	36.662	24.469	14.556	7.245	3.040	1.371
5.2	12.016	6.726	11.158	35.714	26.003	14.274	7.405	3.603	1.843
5.3	12.196	6.739	9.590	36.164	26.656	14.233	7.616	3.742	1.999

Network size (n) = 100,000									
β	Overall		Fraction of posts replicated to between x-y% of the network						
	Mean	Std Dev	0-5	5-10	10-15	15-20	20-25	25-30	>30
14.6	12.139	5.862	9.729	30.792	30.147	18.864	7.632	2.355	0.481
14.7	12.269	5.946	9.540	30.666	29.755	19.029	8.036	2.398	0.576
14.8	12.622	6.051	8.405	30.013	29.678	19.519	8.887	2.725	0.774
14.9	12.899	6.118	8.038	28.069	30.322	20.478	9.259	2.957	0.877
15.0	13.251	6.223	7.159	26.970	30.305	20.705	10.503	3.257	1.100



(a) Mean Buffer Size (b) Max Buffer Size

Figure 3.4: The blog post retrieval accuracy and mean (a) and maximum (b) values of the transfer buffer, $\mathcal{T}$, in a network of 100,000 nodes with a dissemination parameter value of $\beta = 14.8$.

Table 3.9: Replication rate (% of network), blog post retrieval accuracy, and keyword search accuracy as a function of dissemination parameter value of $\beta = 15.7$ and various maximum transfer buffer sizes, $|\mathcal{T}|_{max}$.

$\vert\mathcal{T}\vert_{max}$	Replication		Blog Post Retrieval		Keyword Search @ 10	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
∞	15.124	7.003	96.910	2.580	87.010	2.778
20	12.233	6.342	94.490	2.947	84.069	3.032
25	**12.882**	**6.546**	**95.126**	**2.731**	84.982	2.505
30	13.190	6.568	95.421	2.650	85.396	2.467

iterations later after the spike in the post creation rate. This is due to the extra newly created posts spreading through the network and making their way into the buffers of the nodes they have replicated onto. Similarly the buffer size decreases a few iterations after the lull in the post creation rate. The mean buffer size ranges from 1 to 15 posts and the maximum buffer size ranges from approximately 45 to 450 posts. This maximum buffer size maybe too large as it may place too large a burden on any node uploading or downloading this many posts. We, therefore, repeated our simulations with a maximum transfer buffer size, $|\mathcal{T}|_{max}$, where the posts are selected into $\mathcal{T}$ from the storage area using the same probabilistic method until $|\mathcal{T}| = |\mathcal{T}|_{max}$.

Table 3.9 illustrates the effect of an upper-bound on the size of $\mathcal{T}$ on post replication, retrieval and keyword search. We can achieve the required 95% accuracy with $|\mathcal{T}|_{max} = 25$. However we need to increase the dissemination parameter, β from 14.8 to 15.7. Figure 3.5 shows the post creation rate, retrieval accuracy, and the mean buffer size for a network of 100,000 nodes, $\beta = 15.7$, and $|\mathcal{T}|_{max} = 25$.

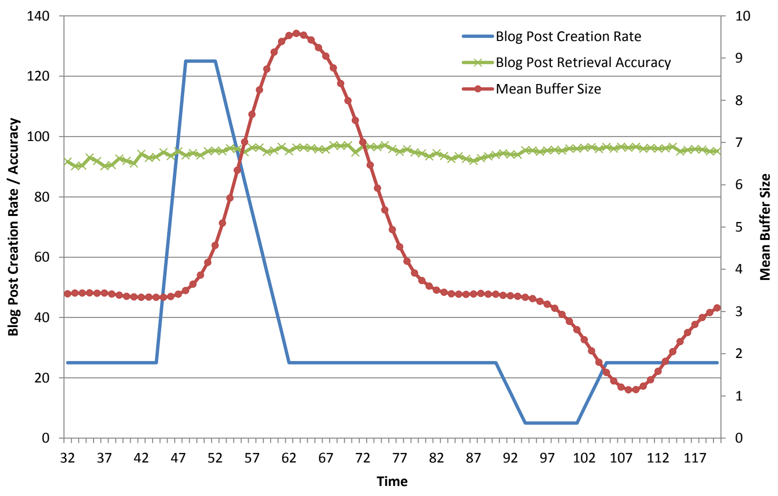


Figure 3.5: The accuracy and mean values of the transfer buffer, $\mathcal{T}$, in a network of 100,000 nodes and a dissemination parameter value of $\beta = 15.7$ with a maximum transfer buffer size of $|\mathcal{T}|_{max} = 25$.

The figure demonstrates that setting a maximum buffer size does not impede the ability of the system to respond to a spike or a lull in the post creation whilst maintaining the required 95% accuracy.

Figure 3.6 shows the distribution of $|\mathcal{T}|$ for $|\mathcal{T}|_{max} = 25$ and $\beta = 15.7$. The two histograms of buffer sizes are averaged over all iterations of all simulations for the two posting rates of 25 and 125 posts per second. When the post creation rate is 25 posts per second, $|\mathcal{T}| \leq 3$ for approximately 80% of the network. In fact, nearly 37% of the nodes in the network have a buffer size of 0, as they are *stifling* all the posts which are eligible for selection into the transfer buffer. Even when the post creation rate spikes to 125 posts per second, 8% of the nodes have a transfer buffer size of 0, and almost 50% of nodes have a transfer buffer size $|\mathcal{T}| \leq 5$.

3.4.2 Churn

Churn is an important factor in any P2P application. Stutzbach and Rejaie [SR06] provide a comprehensive overview of churn in various types of P2P networks. Typically, a node may join the network for periods (mean session lengths) from 1 minute to 1 hour, depending on the type of P2P application, e.g. Gnutella, BitTorrent. Further, it has been observed that the network generally consists of a small portion of highly stable nodes with the remaining peers exhibiting high turnover with session times following an exponential distribution.

We tested our framework with a high level of churn by setting the mean of the session time exponential distribution to just 60 seconds. We initiated a network of 100,000 nodes with a maximum transfer buffer size $|\mathcal{T}|_{max} = 25$, a creation rate of 25 posts per second, and 50,000 active nodes. Churn was then induced into the simulation. New nodes join with a Poisson arrival rate to balance out the exponential exit rate [TKLB07b]. Though the number of active nodes at any given moment does not vary by much, approximately 1,000 nodes exit and join the network at every second.

Table 3.10 illustrates the effect of churn on the accuracy of followed post retrieval and keyword

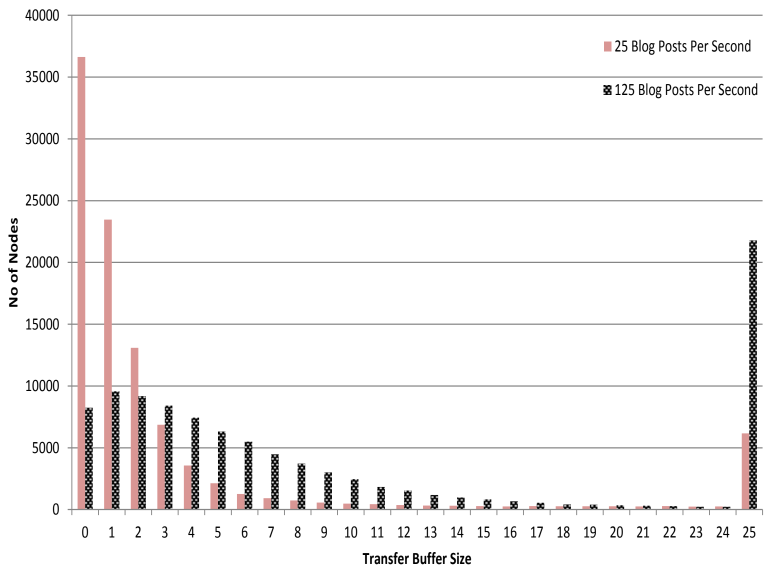


Figure 3.6: The distribution of the size of the transfer buffer, $|\mathcal{T}|$, in a network of 100,000 nodes and with a dissemination parameter value of $\beta = 15.7$ and a maximum buffer size of $|\mathcal{T}|_{max} = 25$.

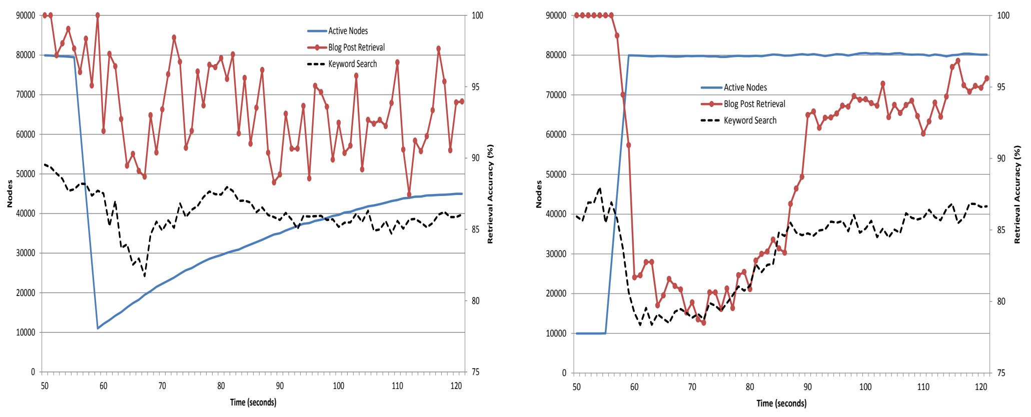
search. As expected, the net effect of churn is to reduce the post retrieval accuracy as well as the the keyword search accuracy. The dissemination of the posts with $\beta = 15.7$, which was previously sufficient to obtain our desired retrieval accuracy (Table 3.9), is now reduced by heavy churn. However, even under heavy churn, the average post retrieval accuracy is 92.53% and keyword search accuracy @ 10 is 84.2% for $\beta = 15.7$. Of course, we can compensate for the effects of churn by increasing the replication rate of posts or by querying more nodes in the network.

We can observe from Table 3.10, that increasing the value of the dissemination parameter, β, from 15.7 to 17.3 restores the accuracy to desired levels.

Figures 3.7(a) and 3.7(b) illustrate the effect of massive exits, i.e. the number of active nodes in the network drops from 80,000 to 10,000, and massive joins, i.e. the number of active nodes jumps from

Table 3.10: The transfer buffer size, $|\mathcal{T}|$, blog post retrieval accuracy, and keyword search accuracy @10 as a function of the dissemination parameter, β, for a network of 100,000 nodes under heavy churn and a maximum buffer size of $|\mathcal{T}|_{max} = 25$.

β	$\lvert\mathcal{T}\rvert$		Blog Post		Keyword @ 10	
	Mean	Std	Mean	Std	Mean	Std
15.7	4.757	0.356	92.526	2.682	84.204	2.008
17.2	6.310	0.407	95.367	2.739	87.378	2.526
17.3	**6.374**	**0.494**	**95.460**	**2.554**	**87.470**	**2.515**
17.4	6.490	0.266	95.524	2.655	87.601	2.474



(a) Massive Exits (b) Massive Joins

Figure 3.7: The change in retrieval accuracies due to various types of churn in a network of 100,000 nodes with a creation rate of 25 blog posts per second, dissemination parameter $\beta = 17.3$, and a maximum buffer size $|\mathcal{T}|_{max} = 25$.

10,000 to 80,000 nodes, respectively. The system is less affected by massive exits since the posts can be found in the remaining nodes. However, massive joins significantly degrade retrieval accuracy since nodes start querying newly joined nodes which do not contain the required posts, and hence the retrieval accuracy initially falls sharply. However, accuracy quickly improves as new nodes receive more posts, via the transfer buffer, as a result of being contacted. Accuracy almost totally recovers within a period of 60 seconds.

3.5 Node Communication Load and System Bandwidth

In this Section, we examine the communication load incurred by a peer on the network and the system bandwidth - the traffic generated by the framework per iteration. We utilize the values displayed in Table 3.10 since it allows us to achieve the required accuracy under heavy churn in a network of 100,000 nodes. Each micro-blog post requires 1 KB in disk space (Section 3.2). For a value of the dissemination parameter of $\beta = 17.3$, each request/response contains a minimum of 8 posts (the contents of $\mathcal{T}$ and the node's most recent post). Assuming at-least 50% compression of the posts and other network information, we make a conservative assumption of 8 KB as the total size of the request/response; which includes the transfer protocol headers, the node's most recent post, the contents of the transfer buffer, the list $\mathcal{L}$ in the request or the posts retrieved from the storage area in the response, and any additional information. All information is compressed before being transmitted over the network.

With each node servicing 0.83 request per second (as described in Section 3.2), a packet size of 8 KB corresponds to each node uploading and downloading 6.64 KB per second or approximately 17.2 GB per month. This is acceptable for a personal computer, but is currently not viable for a mobile phone. For mobile devices, one solution is to communicate, via an app, with a computer which is participating as a node in the network.

For a network size of 100,000 nodes, this corresponds to a total network traffic of 0.664 GB per

Table 3.11: Information retrieval and dissemination in networks of 10,000 and 100,000 nodes with nodes querying $z = 50$ peers every $s = 60$ iterations

$n = 10,000$				
β	Retrieval Accuracy		Dissemination (% of network)	
	Mean	Std Dev	Mean	Std Dev
2.3	94.590	2.628	6.037	2.525
2.4	**94.996**	**2.813**	**6.382**	**2.664**
2.5	95.574	2.432	6.710	2.794

$n = 100,000$				
β	Retrieval Accuracy		Dissemination (% of network)	
	Mean	Std Dev	Mean	Std Dev
9.9	94.646	3.669	6.066	3.033
10.0	**94.942**	**3.615**	**6.272**	**3.103**
10.1	95.291	3.828	6.307	3.096

second. Note that this traffic is spread throughout the internet. The total internet traffic in 2009 was approximately 4,630 GB/s [LIJM+10, Min09] and is forecast to grow by 50% each year, primarily due to video traffic. Using the 2009 figures, the traffic generated by a distributed de-centralized micro-blogging social network consisting of 100,000 active nodes would only constitute 0.014% of the global internet traffic. In comparison, in 2011, BitTorrent constituted 17.2% and 28.4% of the total internet traffic in North America and Europe respectively [San11].

We examine the overall system bandwidth in the next chapter. However, the number of requests any given node services, $\frac{z}{s} = 0.83$, is independent of the network size.

3.6 Summary

In this chapter we considered the design of a micro-blogging social network in an unstructured peer-to-peer network. The general problem is one of probabilistically retrieving highly dynamic information, and therefore the solution is applicable to other applications such as indexing dynamic web pages in a distributed search engine or for a system which indexes newly created BitTorrents in a de-centralized environment.

We analyzed the problem as into one of rapid yet restrained dissemination of multiple simultaneous rumours and this problem is solved using our proposed transfer buffers. Simulations of a 100,000 node network, supporting a normalized post creation rate of magnitude 10 larger than Twitter, provided empirical support to the theoretical analysis and we were consistently able to achieve the required 95% accuracy. Importantly, our proposed system was able to adapt to spikes in the post creation rate. It

Table 3.12: Information retrieval and dissemination in a network designed for 75% accuracy in networks of 10,000 and 100,000 nodes with nodes querying $z = 50$ peers every $s = 60$ iterations

$n = 10,000$				
β	Retrieval Accuracy		Dissemination (% of network)	
	Mean	Std Dev	Mean	Std Dev
0.8	76.721	5.042	2.824	0.966
0.9	**78.321**	**4.925**	**3.005**	**1.056**
1.0	79.403	4.333	3.144	1.134

$n = 100,000$				
β	Retrieval Accuracy		Dissemination (% of network)	
	Mean	Std Dev	Mean	Std Dev
7.2	75.911	5.465	2.997	1.557
7.3	**76.129**	**5.890**	**3.080**	**1.627**
7.3	76.914	5.601	3.175	1.627

is worthwhile emphasizing that 95% accuracy does not imply that a user will miss 1 in 20 of his/her followed posts. As each node makes a request every s seconds, the accuracy is indeed 95% at the first request, but increases to 99.83% by the second request, and 99.99% by the third request, since making two requests to two sets of z random nodes is the same as making one request to $2z$ random nodes as long as the same information is requested. Note, the second and subsequent requests are not redundant as nodes have to periodically query for new micro-blog posts.

We investigated two methods of constructing the transfer buffer of a node - a fixed size buffer, and one in which blog posts are selected with an exponentially decreasing probability according to their age, controlled by the dissemination parameter, β. In order to avoid overloading the bandwidth of nodes, we set a maximum buffer size of $|\mathcal{T}|_{max} = 25$. Importantly, our proposed system was able to adapt to spikes in the post creation rate and maintain the required accuracy.

The PAC search architecture allows the designer to achieve the same accuracy using different combinations of the nodes queried, z, and the replication ratio, $\frac{r}{n}$, and thus we could have also designed the system where the interval between requests, s was 60 seconds with $z = 50$. In this case, the load on the nodes would be the same ($\frac{z}{s}$), but the posts would only need to be spread to 6% of the network. However, in this case, the posts would only be retrieved once per minute. Table 3.11 displays the dissemination parameter, β required for this configuration. Using the same parameters values for nodes queried and interval between requests, we can design our framework for a lower accuracy, say 75%. In this case the blog posts would only need to be replicated onto 3% of the network. Table 3.12 summarizes this configuration.

In our experiments we utilized BM25 to rank the blog posts matching a query. Performance was lower than expected. However, this was shown to be due to the variation in parameter values of the BM25 algorithm which were estimated based on local rather than global statistics. We utilized BM25 for keyword search as it is an established algorithm which has no query-independent features, and thus the performance of a local search can be compared with the search of the entire network. However, BM25 is not the optimal method for searching very small documents, such as micro-blog posts. For example, if a user issues the query `ucl`, a post which mentions `ucl` twice would have a higher score than a post which only contains `ucl` once. In a micro-blog setting, where posts are usually the size of SMS message, of approximately 140 characters, the term frequency is irrelevant. A better ranking algorithm would perform boolean search, and then rank the matching micro-blog posts using query-independent features such as the popularity of the publisher and whether the publisher is followed by the user issuing the query. Implementing query independent features is an interesting avenue for future work.

We also investigated the effect of churn on the network. In simulations of a high churn rate, e.g. average session time of 60 seconds, some degradation is observed. However this can be compensated for by increasing the value of the dissemination parameter, β from 15.7 to 17.3, thereby increasing the replication rate. We also considered massive exits from and joins to the network. We observed that massive exits had much less affect on accuracy than massive joins.

In our simulations, each user follows 35 other randomly chosen peers, and posts are created randomly over the network and at a constant rate. However, in analysis of online social networks suggests that the number of followers follows a power-law distribution. In the next chapter we investigate whether replicating the posts of high popularity users more than others could reduce the overall bandwidth of the system.

Finally, in Section 3.2, we described the storage area of the nodes, where 900 MB is allocated towards storage of other users' blog posts. If all the nodes in a network of 100,000 nodes, publish blog posts which are disseminated into the network, each node's storage area would contain, on average, 90 blog posts from each users. As new blog posts are published, older blog posts will gradually disappear from the network. Thus, an archiving strategy needs to be considered.

Chapter 4

Bandwidth Optimization

In the previous chapter, we proposed a framework for disseminating blog posts in the network in order to facilitate retrieval with high accuracy. Both dissemination and retrieval incur bandwidth costs. In this chapter, we investigate the optimal replication of data, in the sense of minimizing the overall system bandwidth.

The PAC'nPost framework couples replication with querying, i.e. replication occurs periodically and as part of the query - each node seeks to replicate a portion of its recent blog posts when querying random nodes. This makes the system inflexible. For example, we cannot increase the interval between queries without affecting the replication. It is also impossible to arrive at an optimal replication which minimizes the total bandwidth.

In this chapter, we decouple querying from replication. Consequently, in Section 4.2 we analyze and minimize the overall bandwidth usage of the system. PAC search can also be implemented for non-uniform replication [AFC11], and in these cases, the bandwidth can be reduced. In Section 4.2.1, we investigate whether this, alongside a non-uniform follower distribution, observed in real life online social networks, can be used to reduce the system bandwidth. In Section 4.3, we describe the mechanism for decoupled dissemination and retrieval. In Section 4.4, we present the results of our simulations.

We begin with a discussion on the standard deviation of the blog post replication.

4.1 Blog Post Dissemination

The PAC'nPost framework described in Section 3.3 of the previous chapter allows for (i) quick dissemination of published posts onto random number of nodes and (ii) retrieval of posts with high accuracy, specifically 95%. However, we also observe high standard deviation in the replication of posts (Table 3.8). The study of gossip protocols [Het00, New02, MNP04, NMBM07] has not investigated the standard deviation since the focus is the speed of dissemination on various types of networks (homogeneous graph, random graph, complex networks etc.) and the effect of the node degree distribution on the rumour. Furthermore, when the aim is to spread the rumour to the entire network, the standard deviation is irrelevant.

To recap, in a network of n nodes, there are *susceptibles*, x, who do not know the rumour, *infectives*, y, who know the rumour and infect others, and *stiflers*, z, who know the rumour but do not pass it to

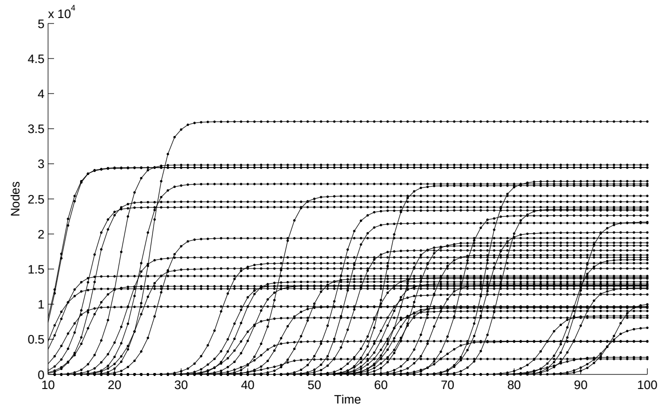


Figure 4.1: The spread of blog posts in a PAC'nPost network of 100,000 nodes

others. Therefore the number of nodes which know the rumour is $y + z$ and $n = x + y + z$. In the PAC'nPost framework, nodes communicate via their transfer buffers, and posts are selected into the buffers with probability $e^{-t/\beta}$. If the post is not selected, it is permanently marked for no further selection at the node, i.e. it is *stifled* at that node. Figure 4.1 shows the replication of 50 randomly chosen posts.

The key difference between the rumour spreading in the PAC'nPost framework and classical rumour spreading is the way in which nodes contact other peers in the network. In classical rumour spreading, infectives contact a set number of other nodes every iteration (also known as an *eager push* [LPR07]), or turn into stiflers with some pre-determined probability. In PAC'nPost, a post is first published it is replicated onto z other nodes, after which it is dependent on being selected into the buffers of nodes which are querying for their followed posts. (Recall, that in the network, roughly $\frac{n}{s}$ query z other random every iteration).

Let's consider the following example: A blog post is published at Node $\mathcal{A}$ and is replicated onto Nodes $\mathcal{B}_1 \ldots \mathcal{B}_z$. Let us now consider the post at Node $\mathcal{B}_1$ at the next iteration with the dissemination parameter, $\beta = 15.7$. Since the post is one iteration old, it has a probability of selection into Node $\mathcal{B}_1$'s buffer of 94%. Let us assume that the post is selected into the buffer of $\mathcal{B}_1$. The best case scenario, with respect to replication, is that this iteration $\mathcal{B}_1$ makes its scheduled request, and contacts z other nodes which do not have a copy of the post, and thus the post is replicated onto z other nodes. The worst case scenario is that $\mathcal{B}_1$ does not make its scheduled request and is not contacted by any other node. It is this selection of post into the buffer and the behaviour of nodes which causes the high standard deviation in the replication of posts.

This process is compounded every iteration until the post is $s_r = 30$ iterations old when it is no longer a candidate for selection into the transfer buffer. However, we must also consider that the nodes use an exponentially decreasing probability of selecting posts into their buffers using the dissemination parameter, β. Posts which are not selected into the buffer at a node are stifled from then on at that node. For $\beta = 15.7$ and $\beta = 10.4$ the probabilities of selection in the buffer when the post is 1 iteration old are 94% and 91% respectively. By the time the post is 15 iterations old, the probabilities of selection

Table 4.1: An example of low, average, and high replication in a network of 100,000 nodes with the dissemination parameter $\beta = 15.7$. At each iteration, the number of susceptibles (Sus), infectives (Inf), and stiflers (Stf) are displayed.

	Low Replication				Average Replication				High Replication			
Time	Total (Inf + Stf)	Sus	Inf	Stf	Total (Inf + Stf)	Sus	Inf	Stf	Total (Inf + Stf)	Sus	Inf	Stf
1	26	99972	26	0	26	99973	26	0	26	99972	26	0
2	52	99948	47	5	52	99948	47	5	127	99873	121	6
3	91	99909	72	19	112	99888	96	16	371	99629	319	52
4	149	99851	107	42	281	99719	215	66	842	99158	646	196
5	247	99753	161	86	526	99474	374	152	1854	98146	1295	559
6	403	99597	232	171	1008	98992	612	396	3397	96603	2083	1314
7	580	99420	292	288	1552	98448	811	741	5404	94596	2854	2550
8	823	99177	359	464	2234	97766	965	1269	7833	92167	3444	4389
9	1125	98875	406	719	3277	96723	1193	2084	10597	89403	3775	6822
10	1461	98539	453	1008	4256	95744	1227	3029	13522	86478	3853	9669
11	1793	98207	398	1395	5480	94520	1325	4155	16823	83177	3818	13005
12	2126	97874	349	1777	6756	93244	1298	5458	19799	80201	3401	16398
13	2385	97615	300	2085	8060	91940	1207	6853	22480	77520	2859	19621
14	2616	97384	243	2373	9025	90975	968	8057	24826	75174	2277	22549
15	2797	97203	193	2604	9873	90127	797	9076	26578	73422	1666	24912
16	2939	97061	143	2796	10606	89394	605	10001	27981	72019	1236	26745
17	3064	96936	99	2965	11182	88818	437	10745	28915	71085	826	28089
18	3137	96863	61	3076	11531	88469	254	11277	29604	70396	533	29071
19	3188	96812	32	3156	11765	88235	159	11606	30114	69886	332	29782
20	3208	96792	22	3186	11901	88099	78	11823	30419	69581	202	30217
21	3235	96765	14	3221	11980	88020	57	11923	30582	69418	112	30470
22	3244	96756	5	3239	12044	87956	47	11997	30682	69318	59	30623
23	3247	96753	2	3245	12095	87905	31	12064	30762	69238	39	30723
24	3248	96752	0	3248	12146	87854	18	12128	30791	69209	18	30773
25	3248	96752	0	3248	12165	87835	12	12153	30805	69195	5	30800
26	3248	96752	0	3248	12180	87820	5	12175	30809	69191	1	30808
27	3248	96752	0	3248	12180	87820	0	12180	30812	69188	2	30810

are 38.5% and 23.6% respectively, *if not already stifled*. Therefore, in PAC'nPost the posts achieve their near-maximal replication value quickly, after which the number of nodes they replicate onto is small.

This can be seen in Figure 4.1, and Table 4.1 shows how a low, average, and high replicating blog spreads through the network. As we can see from Table 4.1, by iteration 18, all three posts have reached 95% of their final replication value, and the divergence in their replication has already occurred.

In the case of a PAC'nPost dissemination, the rumour spreading is finished when either all infectives have turned into stiflers or when the age of the rumour equals s_r. But since we know that the last few iterations achieve very little replication, we can instead focus on the iterations when the post disseminates rapidly. Since each iteration contributes to the standard deviation, we can reduce the standard deviation by reducing the number of iterations when most of the replication occurs. The iterations can be reduced by stifling more, i.e. decreasing the value of the dissemination parameter, β. However this would also reduce the final replication value; and so we need to give our post a bigger initial head start by *seeding* it to more nodes. There we introduce a seeding parameter, z_s, which is the number of random peers a

Table 4.2: The mean and standard deviation of the dissemination in a network of 100,000 nodes with various values of the initial seeding, z_s, and the dissemination parameter, β

z_s	β	Replication	
		Mean	Std Dev
25	15.7	12260.625	5618.118
35	14.3	11800.459	4719.114
35	14.4	12290.276	4818.776
45	13.2	12167.343	4101.417
45	13.3	12321.670	4251.878
55	12.5	12190.358	3771.980
55	12.6	12208.865	4148.011
65	11.9	12100.140	3534.571
65	12.0	12464.873	3644.495
75	11.4	12101.465	3361.568
75	11.5	12412.156	3288.637
85	10.9	12264.088	3290.672
85	11.0	12381.550	3182.490
95	10.5	11870.624	2962.762
95	10.6	12354.760	3049.652
105	10.2	12193.473	2891.419
105	10.3	12421.015	2858.483

Table 4.3: The effect of increased seeding, z_s on replication for the dissemination parameter value of $\beta = 15.7$. For comparison, the replication achieved with $z_s = 100$ and $\beta = 10.4$ is also shown.

z_s	β	Replication	
		Mean	Std Dev
25	**15.7**	**11988.695**	**5811.117**
50	15.7	19622.052	5564.111
75	15.7	25434.505	5570.124
100	15.7	29411.596	5337.038
100	**10.4**	**12033.647**	**2842.767**

node contacts when it publishes a post.

To verify this, we ran 1000 simulations on a network of 100,000 nodes in the PAC'nPost configuration described in the previous chapter, with the additional ability to contact more nodes when the post is published, i.e. $z_s > z$ for various values of the initial seeding, z_s, and the dissemination parameter, β. Tables 4.2 and 4.3 summarize the replication values for the 1,000 simulations. Table 4.2 displays the combinations with which we can achieve the required 12$\%$ replication. With the combination of high stifling and higher seeding we can reduce the standard deviation of replication.

Table 4.3 shows the effect of increased seeding with the same value of β. As expected, this increases the replication. For comparison, the replication of posts with $z_s = 100$ and $\beta = 10.4$ is also shown. From the first and last rows of Table 4.3, we can see that a combination of $z_s = 100$ and $\beta = 10.4$ produces the same replication (12$\%$) as the combination of $z_s = 25$ and $\beta = 15.7$ but with half the standard deviation.

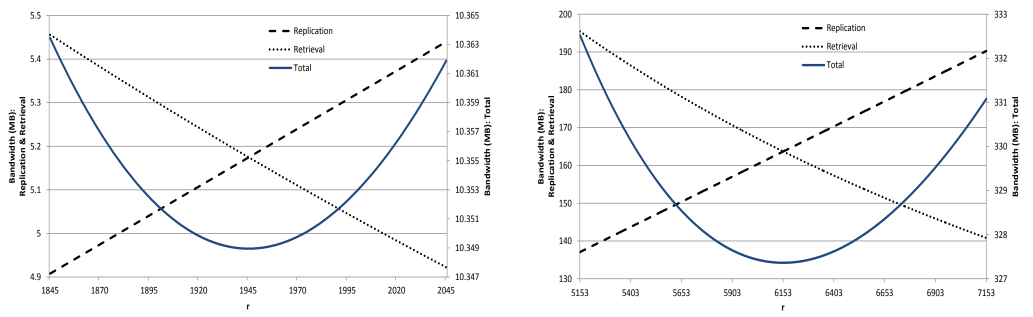
The reduction in the standard deviation is explained by the fact that when we choose high seeding with stronger stifling ($z_s = 100$ and $\beta = 10.4$) we are able to achieve the desired replication in fewer iterations. When we set $z_s = 25$ and $\beta = 15.7$, for most of the posts the last iteration was between 25 and 27 and the mean value of the iteration when the post achieved 95$\%$ of its final replication value was between 18.07 and 18.32. This is detailed in Table A.1. With $z_s = 100$ and $\beta = 10.4$, for most of the posts the last iteration was either 20 or 21 and the mean value of the iteration when the post achieved 95$\%$ of its final replication value was 13.78 and 13.83 for last iteration of 20 and 21 respectively. This is detailed in Table A.2. With higher seeding and stronger stifling, the posts were able to able to achieve 95$\%$ of their final value approximately 4.5 iterations earlier.

Therefore, by increasing both the initial seeding and the stifling, we can disseminate the post faster and in a shorter time period. By reducing the number of iterations when the post replication occurs the most, we can reduce the standard deviation of the dissemination. However, we cannot choose a very large value of the initial seeding, z_s, as one of the assumptions of PAC'nPost is that nodes contact a small number of random peers in order to reduce latency.

4.2 System Design for Bandwidth Optimization

In this section we describe a system where replication is decoupled from retrieval to arrive at the minimum system bandwidth. *The system bandwidth is the traffic generated by the framework per iteration.* The underlying node structure and behaviour is as in the previous chapter - nodes contribute disk space which is utilized for storage and indexing of blog posts, and nodes periodically query a pre-calculated number of random peers to retrieve the followed posts with an accuracy of 95$\%$. A post is given $s_r = 30$ iterations before it is made available for retrieval.

We must first estimate the number of followers (f) a node is likely to have. If Node $\mathcal{A}$ *follows* Node $\mathcal{B}$, then, conceptually, in the *user-follower* graph there exists a uni-directional link from $\mathcal{A}$ to $\mathcal{B}$. Though the total number of follower-links in the Twitter graph is not known, Kwak *et al.* [KLPM10] reported 1.47 billion links for 41.7 million users, and therefore, we set the total number of *follower-links* to $35n$, where n is the number of nodes in the network. For our initial calculations we set $f = 35$ uniformly



$n = 10,000$ $n = 100,000$

Figure 4.2: The bandwidth as a function of replication r for an interval of $s = 60$ seconds, with each node following $f = 35$ other nodes and an accuracy of 95%.

such that each node *follows* 35 others. We relax this restraint when we consider non-uniform replication in Section 4.2.1, and also in our simulations.

Next, we calculate the bandwidth. The calculations are for the PAC'nPost framework, an overlay in which a node can contact another directly, i.e. one hop. For a particular network topology, (e.g. random graph), the bandwidth would need to be multiplied by the average number of hops required for one node to contact another.

When a blog post is created by a node, it is instantly replicated onto r other random nodes. (This is of course idealized, and in Section 4.3 we describe our new method of dissemination). For retrieving the followed blogs, at any given iteration approximately $\frac{n}{s}$ nodes contact z other random nodes.

To calculate the bandwidth we assume the following:

- $c_t = 64$ – the cost, in bytes, when one node contacts another. This is the size of an empty TCP/IP packet, and the cost must be incurred in all requests and responses.
- $c_n = 8$ – the cost, in bytes, of specifying a peer in a request. When a Node $\mathcal{A}$ makes a request to Node $\mathcal{B}$, for each peer Node $\mathcal{A}$ follows, it must specify in the request to $\mathcal{B}$ the peer number, and the number of the last blog post published by that peer which Node $\mathcal{A}$ has already retrieved.
- $c_b = 1000$ – the size, in bytes of a blog post, as described in Section 1.2 of the previous chapter.

We also assume the interval between requests, s, is 60 seconds. We test this new framework with a blog post creation rate of $\alpha = 2.5 \times 10^{-4}$ posts per node per iteration. This is 10 times higher than Twitter's average of 4,600 tweets per second.

As we discussed in Section 2.3 of Chapter 2, the PAC accuracy, $\mu = 1 - \left(1 - \frac{r}{n}\right)^z$ approximates to $1 - e^{-zr/n}$. Let $\epsilon = \frac{zr}{n}$. For $\mu = 95\%, \epsilon = 2.996$. Since a uniform replication of documents in PAC lends itself to a binomial distribution, we can also conceptualize the retrieval process as follows: The accuracy gives us the probability of retrieving *at least one* document and ϵ gives us the number of *expected* documents when we query z random nodes.

The bandwidth ($\mathbf{B}$) has two components: replication and retrieval. The replication bandwidth is a function of the number of blog posts created, αn, and the number of nodes, r, they are replicated onto, and for one iteration can be expressed as

$$\mathbf{B}_r = \alpha n r \left(c_t + c_b\right) \tag{4.1}$$

The retrieval bandwidth has two elements due to the query ($\mathbf{B}_q$) and the response ($\mathbf{B}_e$). Since $\frac{n}{s}$ nodes peform a query at any given iteration, the bandwidth for querying is:

$$\mathbf{B}_q = \frac{n}{s} \times z \times \left(c_t + f c_n\right) = \frac{\epsilon n^2 \left(c_t + f c_n\right)}{sr} \tag{4.2}$$

To calculate the response bandwidth, we need to know the number of expected blog posts that are retrieved by the requests to z random nodes made by a given node, say, Node $\mathcal{A}$. The probability that one of the peers has published a blog post at a given iteration is simply α, and therefore the expected number of blog posts published by the peers that Node $\mathcal{A}$ follows is $fs\alpha$, and the expected number of retrieved blog posts by making a request to z random nodes is $\epsilon fs\alpha$. Thus the bandwidth for replying at any given iteration is given by:

$$\mathbf{B}_e = \frac{n}{s} \times z \times \epsilon fs\alpha \left(c_t + c_b\right) = \frac{n^2 \epsilon^2 f \alpha \left(c_t + c_b\right)}{r} \tag{4.3}$$

The total bandwidth at any given iteration is

$$\mathbf{B} = \mathbf{B}_r + \mathbf{B}_q + \mathbf{B}_e$$

The total bandwidth, $\mathbf{B}$, is a function of the replication rate, r. We can therefore determine the value of r that minimizes $\mathbf{B}$, and then use equation for PAC accuracy to determine the corresponding z for a given accuracy. Figure 4.2 shows the bandwidth as a function of the number of nodes a blog post is replicated onto, r, for an interval of $s = 60$ seconds, an accuracy of 95%, and network sizes of 10,000 and 100,000 nodes. For a network of 10,000 nodes the minimum bandwidth is observed when a blog post is replicated onto 19.45% of the network and nodes query $z = 16$ other randomly selected peers. For a network of 100,000 nodes the minimum bandwidth is observed at 6.153% replication and nodes query $z = 49$ other randomly selected peers.

The overall minimal bandwidth is found when the replication and the retrieval bandwidths are almost the same. As the network size increases from 10,000 to 100,000 nodes, the cost of replicating documents to the same network percentage increases linearly, and the minimal bandwidth is found at a lower replication.

As we can see from Equation 4.2, if we increase the interval between requests, s, we can expect the bandwidth to reduce as well. Table 4.4 shows the optimal replication which gives the minimum bandwidth for various values of s. The Table shows that even if s increases from 30 to 120 seconds, the effect on the overall bandwidth is small. This is because $\mathbf{B}_r$ and $\mathbf{B}_e$ are independent of the query interval, s.

4.2.1 Non-Uniform Replication

We now consider whether replicating and retrieving the blog posts of high-popularity peers separately would decrease bandwidth significantly. Social networks such as Twitter are generally scale-free net-

Table 4.4: The nodes queried, z, the optimal replication, and the associated minimum bandwidth for various values of query interval, s, for an accuracy of 95% in a network of 10,000 and 100,000 nodes where each node follows $f = 35$ other nodes.

$n = 10,000$					
		Optimal	Bandwidth (MB)		
s	z	Replication	Replication	Retrieval	Total
30	15	21.05%	5.599	5.599	11.199
60	16	19.45%	5.174	5.175	10.349
90	16	18.90%	5.027	5.026	10.053
120	16	18.61%	4.950	4.949	9.899

$n = 100,000$					
		Optimal	Bandwidth (MB)		
s	z	Replication	Replication	Retrieval	Total
30	45	6.658%	177.103	177.079	354.182
60	49	6.153%	163.670	163.693	327.363
90	50	5.976%	158.962	158.971	317.932
120	51	5.885%	156.541	156.561	313.102

works which exhibit power law characteristics. In a power law, an entity x is distributed as

$$P(x) = cx^{-\gamma}$$

where c is a normalization constant. In our system, the total number of other peers a node follows is $f = 35$, and thus the total number of edges in our network is $nf = 35n$. Instead of these edges being distributed uniformly, we can construct a power law distribution, with the power law exponent value of $\gamma = 0.68$, such that the top-1% of most popular nodes, cumulatively, account for 20% of the followers. Using this distribution, in a network of 10,000 and 100,000 nodes the most popular node has approximately 6,560 and 26,300 followers respectively. We denote the top-1% most popular nodes as *high popularity* (h), and the rest as *regular popularity* (g) respectively. We then distribute each nodes' followers randomly across the network such that the *following* distribution is similar to the follower distribution (i.e. a small number of nodes follow a large number of peers, but most follow a small number of peers). The result of this transformation of our network is that while the number of peers a node follows is now a power law, 20% of the followed peers are high popularity nodes and the rest are regular popularity nodes.

Another feature of power-law social networks is that a small number of peers account for most of the information created. We, therefore, investigate this by setting the probability of blog post creation by high-popularity peers to various fractions of α. The system now consists of n/s nodes making two sets of requests to z_h and z_g nodes to retrieve blog posts created by high and regular popularity peers

Table 4.5: The minimum bandwidth ($\mathbf{B}_{\mathtt{min}}(h)$ and $\mathbf{B}_{\mathtt{min}}(g)$) achieved via the optimal replication (r_h and r_g) for high and regular popularity peers respectively in a network of 10,000 nodes for a query interval of $s = 60$, and an accuracy of 95%

$p(h)$	High			Regular			Total
$(\times\alpha)$	r_h (%)	z_h	$\mathbf{B}_{\mathtt{min}}(h)$	r_g (%)	z_g	$\mathbf{B}_{\mathtt{min}}(g)$	$\mathbf{B}_{\mathtt{min}}$
0.01	48.02	6	0.255	17.48	17	9.208	9.464
0.1	16.95	18	0.902	17.64	17	8.445	9.347
0.2	13.23	23	1.408	17.85	17	7.597	9.004
0.3	11.73	26	1.872	18.12	17	6.747	8.619
0.4	10.90	28	2.320	18.47	16	5.895	8.215
0.5	10.38	29	2.760	18.95	16	5.040	7.800

Table 4.6: The minimum bandwidth ($\mathbf{B}_{\mathtt{min}}(h)$ and $\mathbf{B}_{\mathtt{min}}(g)$) achieved via the optimal replication (r_h and r_g) for high and regular popularity peers respectively in a network of 100,000 nodes for a query interval of $s = 60$, and an accuracy of 95%

$p(h)$	High			Regular			Total
$(\times\alpha)$	r_h (%)	z_h	$\mathbf{B}_{\mathtt{min}}(h)$	r_g (%)	z_g	$\mathbf{B}_{\mathtt{min}}(g)$	$\mathbf{B}_{\mathtt{min}}$
0.01	15.213	20	8.093	5.531	54	291.281	299.375
0.1	5.366	56	28.549	5.580	54	267.153	295.702
0.2	4.188	72	44.561	5.646	53	240.315	284.876
0.3	3.713	81	59.261	5.731	52	213.434	272.695
0.4	3.451	87	73.442	5.842	51	186.492	259.934
0.5	3.284	91	87.356	5.995	50	159.457	246.814

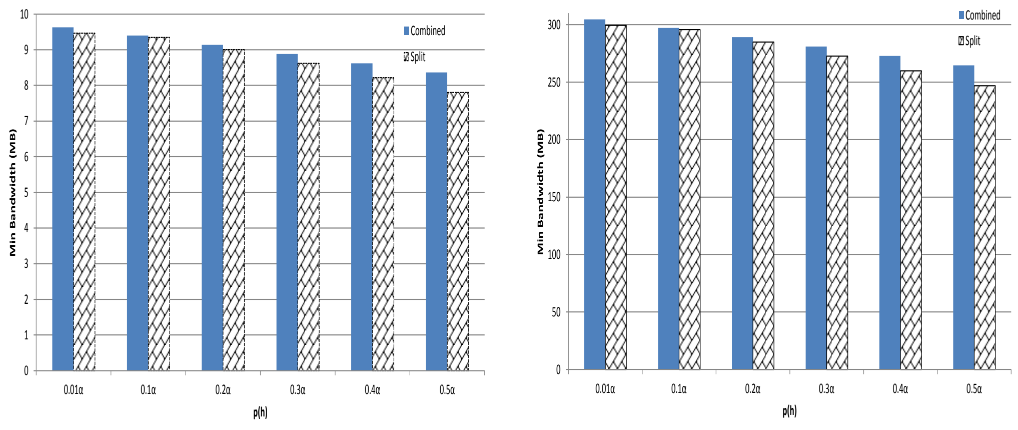
respectively. The high and regular popularity blog posts are replicated onto r_h and r_g nodes respectively, and the minimum bandwidths for high and regular popularity data can be obtained using Equations (4.1)-(4.3), with

$$\mathbf{B}_{\mathtt{min}}(h) \propto p(h) r_h^* \text{ , } \mathbf{B}_{\mathtt{min}}(g) \propto p(g) r_g^*$$

where $p(h)$ and $p(g)$ are the probabilities of blog posts being created by high and regular popularity nodes respectively, with $p(h) + p(g) = 1$. r_h^* and r_g^* denote the optimal replication for minimizing the total bandwidth for high and regular popularity post respectively. The total minimal bandwidth for the entire system is then given by

$$\mathbf{B_{min}} = \mathbf{B_{min}}(h) + \mathbf{B_{min}}(g)$$

Tables 4.5 and 4.6 show the minimum bandwidth ($\mathbf{B}_{\mathtt{min}}$) for various values of $p(h)$ in networks of 10,000 and 100,000 nodes for a query interval of $s = 60$ and an accuracy of 95%. When $p(h) = 0.01\alpha$,



$n = 10,000$ $n = 100,000$

Figure 4.3: The comparison of bandwidths in a non-uniform system between replicating high and regular popularity blog posts the same amount (*Combined*) and replicating high and regular popularity blog posts separately (*Split*) for various rates of creation of blog posts by high popularity peers in networks of 10,000 and 100,000 nodes for an accuracy of 95%

the probability of blog post creation is uniform, (i.e. a high popularity peer is equally likely to create a blog post as a regular popularity peer). When $p(h) = 0.5\alpha$, 50% of the blog posts are being created by the 1% high-popularity peers.

From Table 4.4, we can see that with only uniform replication, the minimum bandwidth is 10.349 MB and 327.363 MB for networks of 10,000 and 100,000 nodes respectively for a query interval of $s = 60$. Of course, the system described in Table 4.4, does not account for non-uniform follower distribution, or for non-uniform blog post creation. Tables 4.5 and 4.6 enumerate the minimum bandwidth when high and regular popularity blog posts are replicated separately. We can compare these figures with the optimal replication of a uniform system (Figure 4.3).

The gains in bandwidth are small. This is due to the nature of the a system in which the data is highly dynamic. Unlike distributed web search where non-uniform distribution of documents can give large reductions in bandwidth [AFC11], peers in a micro-blogging social network *periodically* query for both high and low popularity documents (posts). Furthermore, unlike a relatively static system, we periodically incur the replication and retrieval cost. This erodes any advantage of replicating high popularity nodes to a larger number of nodes.

4.3 Data Dissemination

In this section we describe the mechanism to implement a PAC'nPost variant in which replication and retrieval are decoupled. To attain our goal of rapid-yet-restrained dissemination, when a node publishes a blog post, it immediately replicates its post to z_s nodes. This is the initial *seeding* of the post. At every iteration, a fraction of the network is selected as *Replicators* ($\mathcal{R}$). A node is selected into set $\mathcal{R}$ with probability $e^{-(t-1)/\delta}$ where t is the age of the most recent blog post found in its storage area. I.e. if a node has a blog in its storage which is one iteration old, the node is selected into $\mathcal{R}$. If a node does not have a blog post which is one iteration old, but *does* have a post which is 2 iterations old, it is selected into $\mathcal{R}$ with probability $e^{-1/\delta}$ and so on.

Each replicator must select blog posts to replicate. As with the transfer buffer mechanism described in the previous chapter, at every iteration, each node in the network constructs a *transfer buffer* ($\mathcal{T}$), which consists of a small number of blog posts it has recently come across. Since each post is allowed $s_r = 30$ iterations to spread in the network, we only consider posts which are younger than 30 seconds. Posts are selected into the buffer with an exponentially decreasing probability which is a function of the age of the post: $P(sel) = e^{-t/\beta}$, where t is the number of seconds a blog post has been in existence and β is the dissemination parameter. The transfer buffer has a maximum size, $|\mathcal{T}|_{max}$.

The higher the value of β, the higher the probability that an older post will be selected into the node's transfer buffer and hence the larger the expected size of $\mathcal{T}$. As previously, once a post is not selected into the transfer buffer of a node, it is permanently set for no further selection at that node even if its age is less than s_r seconds, i.e. the node stifles the blog post. At each iteration, every replicator disseminates the posts in its transfer buffer to z_r randomly chosen nodes in the network.

By introducing replicators, we mimic the *eager push* concept in gossip protocols. However, since we need to facilitate multiple simultaneous rumours, we must utilize the transfer buffers mechanism. By *stifling* older blogs, we can achieve restrained dissemination. Parameter β keeps older blogs from being selected into $\mathcal{T}$. Parameters z_s (initial seeding) and δ control the number of replicators active at any iteration and z_r controls the amount of dissemination the replicators are able to accomplish. Many combinations of z_s, δ, and z_r can give us our required replication; we can have fewer replicators contacting a larger number of nodes (z_r) or a larger number of replicators contacting fewer nodes.

Table 4.7: The bandwidth, replication, and retrieval accuracies in networks of 10,000 and 100,000 nodes with blogs seeded to $z_s = 50$ nodes, replicator selection parameter $\delta = 0.66$, and a maximum transfer buffer size of $|\mathcal{T}|_{max} = 100$

Network size $(n) = 10,000$, transfer buffer parameter $\beta = 6.9$											
z_r	Bandwidth (MB)		Replication (% of network)		Blog Post Retrieval		Keyword Search					
							@ 10		@ 20		@ 30	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
8	8.320	0.841	18.879	4.692	93.851	3.736	88.009	4.088	90.858	3.721	91.594	3.911
8.5	**9.209**	**1.003**	**21.284**	**4.978**	**95.542**	**2.555**	89.143	2.751	91.882	2.617	92.574	2.505
9	9.758	1.090	24.704	5.685	96.654	3.319	89.957	3.091	92.618	2.994	93.200	3.473

Network size $(n) = 100,000$, transfer buffer parameter $\beta = 15.9$											
z_r	Bandwidth (MB)		Replication (% of network)		Blog Post Retrieval		Keyword Search					
							@ 10		@ 20		@ 30	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
7	267.288	6.752	5.564	1.708	91.992	1.236	84.748	1.289	86.452	1.143	87.91	1.207
7.5	**307.320**	**7.043**	**6.571**	**1.961**	**95.845**	**1.009**	88.506	1.010	90.443	1.063	91.617	1.030
8	349.106	8.665	7.309	2.097	97.924	0.603	90.599	0.598	92.512	0.549	93.655	0.629

4.4 Simulations

To verify our framework in which replication and retrieval are now separate, we performed simulations based upon the parameters discussed in the previous section.

As described in Section 4.2.1, the total number of *follower* links in the network is $35n$ and is distributed via a power law such that the top-1% most popular nodes have 20% of all the followers. These top-1% of nodes also account for 20% of all blog posts created. This is motivated by the fact that in Twitter a small proportion of users are responsible for a disproportionately large amount of Tweets and the larger the number of followers a user has, the more they are likely to Tweet [Sys09]. We tested our simulations with a blog post creation rate 10 times larger than Twitter. This corresponds to 2.5 and 25 posts per iteration in a network of 10,000 and 100,000 nodes respectively. Nodes query every $s = 60$ seconds to retrieve the blog posts they follow by querying $z = 16$ and $z = 49$ random nodes in networks of 10,000 and 100,000 nodes (Table 4.4). Since we randomize the request time at each node, there are roughly $\frac{n}{s}$ nodes making a request at any given moment.

As with the set of simulations in Chapter 3, the TREC Micro-blog Tweets2011 corpus was used as the source for posts and keyword search was performed alongside every query for retrieving followed blog posts,[1] and BM25 was utilized as the retrieval model.

We tested various values of blog seeding (z_s), replicator selection (δ), and transfer buffer construction (β). Our aim is to replicate the blog posts to 19.45% and 6.153% of the nodes in networks of 10,000 and 100,000 nodes respectively and to compare the predicted bandwidth and retrieval accuracies against their observed values. We performed 10 simulations for each value of the parameters and network size (10,000 and 100,000 nodes). Each simulation lasted for 1200 iterations, so that each node in the network would make at-least 20 *requests*. At each iteration of each simulation, we recorded the total bandwidth generated, and the blog posts retrieved by the querying nodes. At each iteration, for every node which made a *request*, we recorded the accuracy for both post retrieval and keyword search. The accuracy values reported are averaged over all iterations of all simulations of each blog post replication and network size (n) combination. To calculate the dissemination of posts, we recorded each post's replication at the end of each simulation.

Table 4.7 summarizes the simulations for network sizes of 10,000 and 100,000 nodes. The values are obtained by seeding each blog to $z_s = 50$ nodes, setting the replicator selection parameter to $\delta = 0.66$. The table shows the effect of increasing the value of the nodes contacted by the replicators (z_r) on the dissemination and consequently the retrieval accuracies. As expected, as the replicators contact more nodes z_r, the blog posts are replicated onto more of the network. As the replication increases, nodes are able to retrieve the blog posts they follow with higher accuracy by querying the same number of random nodes. Other combinations of these parameter values can also yield the same mean replication but with nearly identical bandwidth and standard deviation.

As with the original framework in Chapter 3, we found that the accuracy of keyword search is lower than post retrieval (following) accuracy, due to BM25 having access to only the node's local statistics.

[1]Note, the bandwidth generated by the keyword search is not taken into account in Table 4.7.

Table 4.8: Keyword search accuracy when each node has access to global BM25 parameter values in a network of 10,000 nodes

z_r	Keyword Search					
	@ 10		@ 20		@ 30	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
8	94.326	2.968	94.963	2.753	94.982	2.743
8.5	95.374	3.635	95.992	3.649	96.089	3.451
9	95.889	2.454	96.508	2.342	96.586	2.580

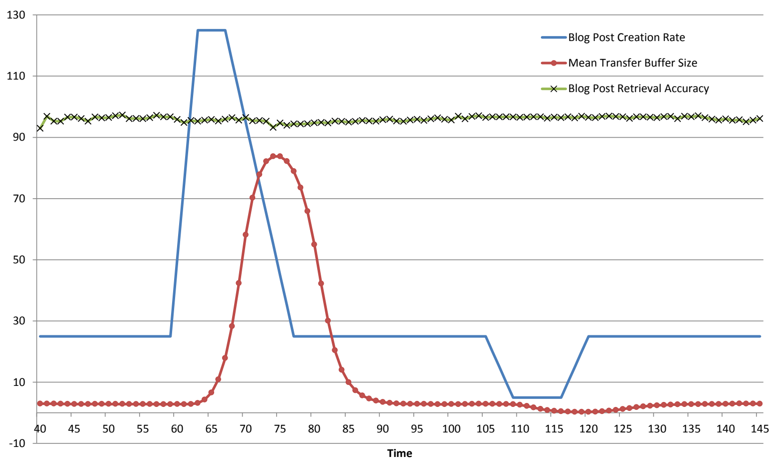


Figure 4.4: The blog post retrieval accuracy and mean values of the transfer buffer, $\mathcal{T}$, in a network of 100,000 nodes, with $\beta = 15.9$, $|\mathcal{T}|_{max} = 100$, $z_s = 50$, $\delta = 0.66$, and $z_r = 7.5$ nodes. The vertical axis shows the number of blog posts for the blog post creation rate, the mean transfer buffer size and the percentage accuracy for retrieval.

Table 4.8 shows the keyword accuracies with the global BM25 parameter values made available at the nodes.

We also tested the new decoupled system by inducing a spike in the blog post creation rate. For completeness, we also introduced a lull period where the post creation rate drops to a fifth of the normal post creation rate. Figure 4.4 shows the blog post retrieval accuracy and the mean transfer buffer size for a network of 100,000 nodes when a spike and a lull is induced in the post creation rate. The figure demonstrates that the retrieval accuracy remains at around 95% regardless of the blog post creation rate, as the transfer buffers grow and constrict automatically.

Finally for completeness, we subjected the new framework to the three types of churn. First, we set the mean of the session time exponential distribution to just 60 seconds, and initiated a network of 100,000 nodes with 50,000 active nodes. Churn was then induced into the simulation. New nodes join with a Poisson arrival rate to balance out the exponential exit rate. As before, approximately 1,000 nodes exit and join the network at every second. We ran two sets of churn simulations - ones in which the high popularity nodes were designated as the stable nodes, and ones in which the stable nodes were chosen at random. The results were nearly identical.

Table 4.9: The blog post retrieval accuracy, and keyword search accuracy @10 for various replicator contact rate, z_r, in a network of 100,000 nodes under heavy churn.

	Blog Post		Keyword @ 10	
z_r	Mean	Std	Mean	Std
7.5	91.265	1.482	81.204	1.806
8.0	93.381	1.339	84.878	1.626
8.5	**95.660**	**1.454**	**88.270**	**1.815**

Table 4.9 illustrates the effect of churn on the accuracy of followed post retrieval and keyword search. As expected, the net effect of churn is to reduce the post retrieval accuracy as well as the the keyword search accuracy. Of course, we can compensate for the effects of churn by increasing the replication rate of posts or by querying more nodes in the network. We can observe from Table 4.9, that increasing the number of nodes which are contacted by the replicators, z_r, from 7.5 to 8.5, restores the accuracy to desired levels.

We subjected the new decoupled framework to massive exits, i.e. when the number of active nodes in the network drops from 80,000 to 10,000, and massive joins, i.e. the number of active nodes jumps from 10,000 to 80,000 nodes, respectively. We found that the effect of massive exits and joins on new framework was similar to the original PAC'nPost framework with massive exits not degrading the accuracy as posts can be found in the remaining nodes. Similarly, massive joins temporarily degraded accuracy as newly joined nodes were queried for posts which they did not have have. However, accuracy quickly recovers as new nodes receive more posts, by being contacted by the replicators.

4.5 Summary

In this chapter we first investigated the large value of the standard deviation of blog post replication in PAC'nPost. Unlike classical rumour spreading, posts in PAC'nPost are copied via the transfer buffers of nodes making their scheduled request for followed posts. We showed, via simulations, that increasing both the initial seeding and the stifling can spread the post faster but in fewer iterations, thereby reducing the standard deviation of the dissemination.

Instead of each node following 35 other peers, we investigated a non-uniform distribution where the top-1% account for 20% of all followers in the system. We analyzed the overall bandwidth generated by our framework and we derived that a network of 10,000 nodes will produce a minimum bandwidth of 9 MB per iteration when posts are replicated onto approximately 20% of the network. In a network of 100,000 nodes, a minimum bandwidth of approximately 300 MB per iteration is generated when posts are replicated onto approximately 6% of the network.

We investigated whether replicating the posts of high popularity users more than the posts of regular popularity users can be used to reduce the bandwidth. Unlike a distributed web search scenario, in a P2P micro-blogging social networks nodes periodically query for both high and regular popularity content, and thus the reduction is minuscule.

In order to decouple replication from retrieval, we introduced a set of nodes, known as replicators. These replicators mimic the *eager push* mechanism in gossip protocols, and via the selection of posts in the transfer buffer, can facilitate multiple simultaneous rumours in a rapid-yet-restrained manner. We subjected our framework to spikes in the data creation rate and to various type of churn.

In the next chapter, we examine some trending keywords in a micro-blogging social network.

Chapter 5

Retrieval of Trending Keywords

In this chapter we investigate the problem of identifying trending information in a peer-to-peer micro-blogging social network. Whilst identifying trending topics in a centralized system is relatively straightforward, in a peer-to-peer environment the participating nodes do not have access to global statistics such as the micro-blog post creation rate and the frequencies of the keywords. We propose a two step approach - first nodes make an estimate of the frequency of the topic within the network, and then they ensure that the topic has appeared in blog posts which were created at different times within a specified interval.

An important feature of micro-blogging social networks is trending information. Trending topics are valuable since they often reflect a news-worthy event. Whereas a user would only receive updates from other peers he or she follows, a trending topic, which could loosely be defined as a keyword which is posted at a greater rate than other keywords within a given time period, is displayed to all users. Whilst some trending topics tend to be banal or whimsical, trending topics often include important early indication of breaking news which then invites users to search on the topic. Consequently, there is a self-reinforcing effect of the trending topic, as users who have performed searches add their own posts on that topic.

In centralized systems, such as Twitter, a trending keyword is generally defined as a topic that is tagged at a greater rate than other tags[1]. The retrieval of trending keywords in P2P micro-blogging network can be conceptualized as follows: Given a post creation rate of α posts per iteration, we define a *trending keyword* as one which exists in a proportion of α for t consecutive iterations. Let the trending keyword appear in a fraction y of the post creation rate, α. If a node receives posts containing the trending keyword such that at least one post belongs to each of the t iterations (or a large fraction of the t iterations), then the node can recognize the keyword as trending without any central coordination.

In the system described in Section 3.3[2], the restrained replication allows a post to be replicated onto r/n fraction of the network. Similarly, a node is likely to receive $\frac{\alpha r}{n}$ of the posts and $\frac{\alpha y r}{n}$ of the posts containing a trending keyword at any given iteration. Over t iterations, the probability it receives i posts

[1] `https://support.twitter.com/entries/101125-about-trending-topics`

[2] This work was completed in the framework described in Chapter 3. However, it is valid for the bandwidth optimized version of PAC'nPost as well, since it piggybacks on the scheduled query for the retrieval of posts.

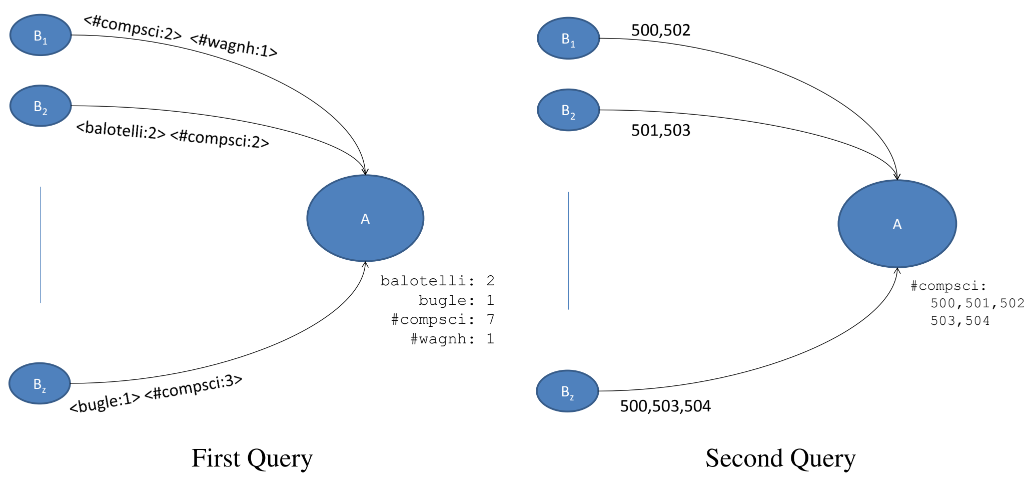
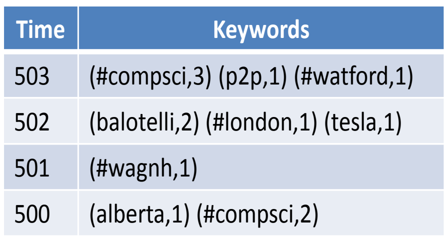


Figure 5.1: An example of the 2-step method. Keyword #compsci is trending in the network between iterations 500-504. The first query allows Node A to sum up the frequencies of the keywords to select keywords for the second query. The second query returns the timestamps of keyword #compsci only.



Time	Keywords
503	(#compsci,3) (p2p,1) (#watford,1)
502	(balotelli,2) (#london,1) (tesla,1)
501	(#wagnh,1)
500	(alberta,1) (#compsci,2)

Figure 5.2: An example of the keyword storage. Keywords are extracted from posts, and stored in the set which corresponds to the iteration at which the post was published.

containing the trending keyword is simply

$$p(i) = \binom{t}{i} \left(\frac{\alpha yr}{n}\right)^i \left(1 - \frac{\alpha yr}{n}\right)^{t-i}$$

and the probability that the node will come across the trending keyword at t, $t-1$, or $t-2$ times is very low.

To facilitate the identification of trending keywords, nodes are required to implement a *keyword storage* area which stores keywords of the recent blog posts the node has encountered. In our framework, the nodes receive blog posts by contacting z other nodes or by being contacted by a peer making its scheduled query for micro-blogs it is following. Keywords are extracted from these posts and inserted into sets where each set in the keyword storage corresponds to one iteration (Figure 5.2). We can utilize the information from the keyword store alongside the scheduled exchange of transfer buffers, to determine whether a keyword is trending in the network.

One rudimentary method would be to exchange all the contents of the j most recent sets when peers contact each other as part of the scheduled query mechanism. As we increase the value of j, a trending keyword gets more time to replicate onto more nodes, and within a short period of time, each node can observe the keyword occurring in t consecutive iteration. However, this method consumes a very large amount of bandwidth - 10 to 15 times the size of the transfer buffer depending on the value of j, since it amounts to an non-stifled exchange of data between nodes.

To observe a keyword as trending in an unstructured P2P network, the keyword information is instead exchanged as follows:

- When Node $\mathcal{A}$ makes its scheduled query to z random nodes, $\mathcal{B}_1$ to $\mathcal{B}_z$, each of the z nodes computes the cumulative frequency of each keyword found in the $(s + t)$ most recent sets in the keyword storage, and returns the top-j keyword-frequency pairs, where j is a system parameter. Since s is the number of iterations required for replicating a blog in the network, and t is the minimum number of consecutive iterations a keyword must appear in the published posts, the nodes are only required to search within the $(s + t)$ most recent sets. By restricting the responding nodes' search of their keyword storage in this manner, we avoid searching for the same keywords in the future, and reduce the bandwidth.

- Node $\mathcal{A}$ combines the results from nodes $\mathcal{B}_1$ to $\mathcal{B}_z$. The frequencies are added for each keyword. For example, if Node $\mathcal{B}_1$ returned `[(#compsci,5);(balotelli,2);(p2p,1)]` and $\mathcal{B}_2$ returned `[(#compsci,4);(data,1)]`, Node $\mathcal{A}$ will have a combined sorted list of `[(#compsci,9);(balotelli,2);(data,1);(p2p,1)]`.

- When Node $\mathcal{A}$ makes its next scheduled query s seconds later, it once again contacts z random nodes, and requests, the publication times (iteration numbers) of the keywords whose frequencies are w times or greater than the median frequency of the keyword-frequency list it obtained via its previous query. The constant w is a system parameter.

- The z nodes return the times when the requested keywords were encountered. These are then merged and sorted at Node $\mathcal{A}$. For example if Node $\mathcal{A}$ had requested the times for `#compsci`, and received [($\mathcal{B}_1$:500,503);($\mathcal{B}_2$:500,501);($\mathcal{B}_2$:501,502,503)],
Node $\mathcal{A}$ can infer that the keyword `#compsci` was trending during times `500-503`.

By summing the frequencies of the keywords, we leverage the *sample index* of a PAC based system as described in Section 2.3. The size of the sample index is the number of non-unique documents indexed by the sample of z nodes. The size of sample index for a system designed for 95% retrieval accuracy is approximately 3 time the number of unique documents in a collection. Thus, the observed value of the total frequency should be 3 times its actual frequency. A detailed analysis of the sample index can be found in [CFH09].

In our proposed method, at each iteration, when nodes make their scheduled request, they request the iteration numbers of the most frequent keywords in the keyword-frequency list obtained in their previous query, and receive a new keyword-frequency list which they will utilize in their next scheduled query. This process piggy-backs on the already scheduled request so no new requests are made by the nodes. However the size of the requests and responses is increased, with a corresponding increase in bandwidth, as discussed shortly.

Once a node has merged and sorted the retrieved times (measured in iteration number), it records the first time the keyword is observed and then counts the number of times the keyword is observed in the subsequent t iterations. The ratio of this number to t is defined as the *trending fraction*, σ. If σ has a high value at a node, the node can infer that the keyword has occurred in most, if not all, of the t iterations and thus can be displayed to the user as a trending keyword.

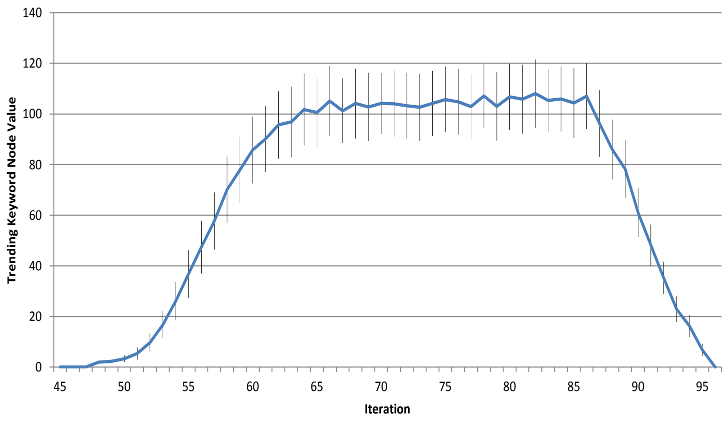


Figure 5.3: The frequency of occurrence of the trending keyword as observed at nodes in the network when the keyword occurs in $y = 15\%$ of the posts created between iterations 46 and 55.

5.1 Simulations

To verify our proposed method, we simulated the system described in Section 3.4. The network consisted of 10,000 nodes. The dissemination parameter, $\beta = 5.1$, which resulted in blog posts being replicated to, on average, 12% of the network (as detailed in Table 3.6). Each node made a request to $z = 25$ other random nodes every $s = 30$ iterations.

At each iteration, we created $\alpha = 25$ micro-blog posts at random nodes in the network. All posts were sampled without replacement from the TREC Micro-blog Tweets2011 corpus. For $t = 10$ consecutive iterations, we injected a trending hashtag into y percent of the posts created at that iteration. We set $w = 10$ so that nodes request the times (iterations) of keywords whose frequency is 10 times or higher than the median frequency. We ran 10 simulations for each value of y and j and experimental results report the average of the 10 simulations.

Figure 5.3 shows the mean values over 10 simulations of the frequency of the trending keyword with $j = 3$ and $w = 10$ where a trending keyword appeared in $y = 15\%$ of the posts created between iterations 46 and 55. In each simulation, of the 250 posts created in this time period, 36 posts contained the trending keyword. These posts are replicated by the transfer buffer mechanism. A node making its scheduled request to z other peers will receive the top-j keyword-frequency pairs from each of the z nodes' keyword storage. The node then accumulates all the frequencies for each keyword. Figure 5.3 shows, for example, a node making its scheduled request at iteration, say, 70 would find the accumulated frequencies of the trending keyword to be 105, which is 3 times the total number of times the trending

Table 5.1: The number of nodes with the associated trending fraction, σ, and the percentage increase in bandwidth, ($\mathbf{Bw}$), for y, the percentage of posts per iteration in which the trending keyword is inserted.

$j = 1$			
	y		
	5%	10%	15%
$\sigma \geq 80\%$	9234	9989	9990
$\sigma \geq 90\%$	8074	9972	9989
$\mathbf{Bw}$ (%)	6.44	9.37	12.39
$j = 2$			
	y		
	5%	10%	15%
$\sigma \geq 80\%$	9379	9999	10000
$\sigma \geq 90\%$	8204	9987	9998
$\mathbf{Bw}$ (%)	9.73	12.92	15.82
$j = 3$			
	y		
	5%	10%	15%
$\sigma \geq 80\%$	9759	9999	10000
$\sigma \geq 90\%$	9162	9995	10000
$\mathbf{Bw}$ (%)	13.84	17.15	19.58

keyword has been published, as predicted by the *sample index*. A node making its scheduled query at iteration 58, would calculate the trending keyword frequency as less than 3 times the actual value. This is due to the fact that the posts containing the trending keyword have not yet being fully replicated. The node would make another request $s = 30$ iterations later when it would be able to calculate the total frequency of the trending keyword correctly.

Table 5.1 displays the number of nodes with trending fractions greater than 80% and 90% for various values of y and j, the number of keyword-frequency pairs returned by queried nodes. When a trending keyword appears in $y = 5\%$ of the posts, with nodes returning the top-2 keyword-frequency pairs, i.e. $j = 2$, we observe that 9,379 nodes of a total of 10,000 nodes, have a trending fraction of 80% or higher. Thus, a very high fraction of nodes, 93%, can infer that the keyword is trending in the network.

Table 5.1 also shows the increase in bandwidth caused by j and y. In order to facilitate trending keywords, nodes transfer *tokens*, i.e. keywords, frequencies of keywords, and iteration numbers. We sum up the number tokens transferred between nodes for each iteration of each simulation. We then take the maximum value and divide by the number of nodes to give an average number of tokens that each node needs to accommodate alongside its transfer buffer. This number of tokens per node per iteration is the extra bandwidth generated for trending keywords. The percentage increase is calculated against the expected bandwidth of 15 words per blog post[3] and an an average of 3 blog posts in the transfer buffer.

Both j and y contribute to the bandwidth generated for trending keywords. As j increases, the number of keyword-frequency pairs increase, and as y increases, the keyword is replicated onto more nodes which then return the iteration numbers.

Table 5.2: The number of nodes with a trending fraction of $\sigma \geq 80\%$ in a system with 5 trending keywords for various values of y and j.

	j		
	1	2	3
$y = 5\%$	15	1064	6153
$y = 10\%$	1849	8296	9942
$y = 15\%$	5605	9277	9986
Bw (%)	15.31	30.74	37.73

We also tested the system for multiple trending keywords. We injected 5 trending keywords for $t = 10$ consecutive iterations with 1 keyword appearing in 5%, 3 keywords appearing in 10%, and 1 keyword appearing in 15% of the posts respectively. Table 5.2 displays the number of nodes with $\sigma \geq 80\%$ for the 5 trending keywords. When j, the number of keyword-frequency pairs, is small, the one trending keyword occurring in 5% of the posts is not included in the keyword-frequency pairs due to the presence of keywords which occur in 10% and 15% of the posts. Consequently, with $j = 2$, the keywords occurring in 5% of the posts are not identified as trending by the majority of the nodes. However, Table 5.2 shows that with $j = 3$, the system can accommodate multiple trending keywords.

5.2 Summary

Trending keywords in a micro-blogging social network are useful to users as they provide information which users may not normally receive via the peers they follow. Since they are made available to all users, in a sense, they form a virtual super-node with which all users are connected.

In this chapter, we proposed a two-step algorithm for the retrieval of trending information in a peer-to-peer micro-blogging social network. In the first step, nodes makes an estimate of the frequency of the keywords in the network. In the second step, they select the most frequent keywords and ensure that these keywords have occurred in posts which were published at different times within a given interval.

Our simulations showed that when a system is designed to retrieve one trending keyword, the increase in bandwidth is relatively small (Table 5.1). However, in this configuration, if there happened to be multiple trending keywords, it is likely that most nodes would not identify any of them as trending. For a configuration designed to identify multiple keywords (Table 5.2), the increase in bandwidth, is, unfortunately, significant. Reducing the bandwidth for this configuration is an avenue for future work.

[3]http://web.resourceshelf.com/go/resourceblog/54128

Trending information is susceptible to spam. Since a spammer knows that normal users are likely to search on the trending keyword, the spammer would publish posts containing the trending keyword, which may get retrieved by the user querying the trending keyword. Identifying spammers in P2P micro-blogging is also an avenue for future work.

Chapter 6

Conclusion

In this thesis we developed a framework for micro-blogging in an unstructured peer-to-peer network. The general problem is one of probabilistically retrieving highly dynamic information. We analyzed the problem as that of rapid yet restrained dissemination of multiple simultaneous rumours. This problem is solved using our proposed transfer buffers.

Simulations of a 100,000 node network, supporting a normalized post creation rate of magnitude 10 larger than Twitter, provided empirical support to the theoretical analysis and we were consistently able to achieve the required 95% accuracy. Importantly, our proposed system was able to adapt to spikes in the post creation rate. We demonstrated via our simulations that the nodes' transfer buffers automatically expanded to accommodate the extra volume of posts during a spike and then contracted as the spike subsided. Throughout the spike or a lull in the post creation rate, our framework was able to operate at the required 95% accuracy. We set a maximum size for this flexible transfer buffer and showed that we can maintain the desired accuracy by marginally increasing the value of the dissemination parameter.

Since both replication and retrieval incurs bandwidth costs, we analyzed the overall bandwidth generated by our framework and we derived that a network of 10,000 nodes will produce a minimum bandwidth of 9 MB per iteration when posts are replicated onto approximately 20% of the network. In a network of 100,000 nodes, a minimum bandwidth of approximately 300 MB per iteration is generated when posts are replicated onto approximately 6% of the network.

We also examined trending keywords. Whilst identifying trending topics in a centralized system is relatively straightforward, in a peer-to-peer environment the participating nodes do not have access to global statistics such as the micro-blog post creation rate and the frequencies of the keywords. We proposed a two step approach - first nodes make an estimate of the frequency of the topic within the network, and then they ensure that the topic has appeared in blog posts which were created at different times within a specified interval.

Our framework works in a fully distributed manner without any centralized coordination. Such a decentralized distributed service may prove useful for users if centralized services, such as Twitter, are blocked. It may also be helpful for users who wish to blog anonymously or are concerned about censorship.

In PAC'nPost all published posts are replicated in the network. This means that published posts are available for retrieval even if the publisher goes offline or is forced offline. Furthermore, since users query random nodes to retrieve the posts of users they follow and not directly the node that they follow, we do not punish users for being popular, i.e., all nodes are queried equally and popular nodes are not inundated with requests.

Our framework is also applicable for other applications such as indexing dynamic web pages in a distributed search engine or for a system which indexes newly created BitTorrents in a de-centralized environment. We can envisage a distributed search engine in which a few dedicated nodes are used to crawl and index web pages from a large collection or the world wide web. These nodes can then use PAC'nPost to replicate the index of the web pages onto a predetermined number of nodes via the dissemination parameter. This describes a rough outline of a peer-to-peer distributed search engine, where search could be performed by sending the query to a predetermined random number of nodes.

Another application for our framework could be for publishing and searching for BitTorrents in a P2P network. A node which creates a BitTorrent could replicate the torrent onto other nodes via PAC'nPost, and this torrent can then be made available to other nodes via keyword search. This would eliminate the need for *tracker* websites which are a point of vulnerability for the BitTorrent network.

6.1 Future Work

There are several avenues of future work in PAC'nPost:

- **Archiving**: Nodes operate a First In First Out strategy for storing posts in their storage areas. Since the size of storage area is fixed, eventually newer posts will replace older ones, which will be lost from the network. Thus, an archiving strategy needs to be considered.
- **Trending keywords bandwidth**: In our two-step method for retrieving trending keywords, the increase in bandwidth for retrieving multiple keywords was not insignificant. Reducing this to more moderate level is an avenue for future work.
- **Deletion**: Our framework offers no way of deleting or recalling a post once it has been published. A recall/delete mechanism can be investigated.
- **Recommendation**: In our framework, nodes store a fraction of posts published in the network in their storage areas. These posts can be used to recommend text or multimedia relevant to the user based on their publishing history. The accuracy of such a recommendation could be further improved by making a query to z random nodes and utilizing the sample index to provide better recommendations to the user.
- **Query independent features for ranking**: In our experiments we utilized BM25 to rank the blog posts matching a query. However, BM25 is not the optimal method for searching very small documents, such as micro-blog posts, as we discussed in chapter 3. A better ranking algorithm would perform a boolean search, and then rank the matching micro-blog posts using query-independent features such as the popularity of the publisher and whether the publisher is followed by the user issuing the query. These two metrics could be made available at the node responding to the query

fairly easily. Moreover, we could incorporate further query independent features such as *follower of a follower* to further improve keyword search. For example, if @STUDENT follows @UCLCS and @UCLCS follows @UCL, a post published by @UCL matching @STUDENT's query should be ranked higher.

- **Content pollution**: The framework can suffer from two types of content pollution: (i) users assuming a false identity to spread disinformation, and (ii) users posting spam. The data posted by these malicious users would not appear in the timeline of honest users as the nodes only retrieve blog posts of the peers they follow. However, these malicious posts could appear in the results of keyword searches. One method of combating this form of content pollution is a system of complaints. Users who encounter nefarious blog posts would issue a complaint against the node. This complaint would affect the node's reputation and affect the ranking of the node's posts in future searches. However, the system also needs to consider the reputation of the node making the complaint as it could be a malicious user attempting to poison the reputation of a honest node. Since each node cannot store the reputations of all the nodes in the network, a decentralized reputation mechanism can be explored.

Appendix A

Replication and last iteration values for 1,000 simulations

Table A.1: The last iterations and post replication values for $z_s = 25$ and $\beta = 15.7$

Last	Sims	Replication		93%		94%		95%		96%		97%	
Iter		Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
23	11	4015.36	1296.33	17.00	0.00	17.09	0.29	17.46	0.50	18.00	0.43	18.46	0.50
24	85	6740.66	3669.85	17.14	0.41	17.46	0.57	17.86	0.51	18.18	0.47	18.80	0.50
25	208	9980.38	5054.79	17.35	0.52	17.65	0.59	18.07	0.47	18.42	0.57	19.02	0.50
26	284	13169.76	5993.24	17.53	0.55	17.88	0.57	18.21	0.57	18.64	0.59	19.18	0.57
27	201	12865.51	5249.41	17.65	0.55	17.97	0.52	18.32	0.55	18.76	0.58	19.29	0.55
28	124	13663.19	5077.93	17.75	0.55	18.07	0.59	18.41	0.58	18.90	0.61	19.39	0.61
29	53	14936.93	5487.94	17.76	0.47	18.09	0.49	18.40	0.49	18.94	0.56	19.45	0.60
30	24	13411.58	5153.31	17.71	0.68	18.17	0.47	18.50	0.71	19.00	0.76	19.42	0.76
31	9	15112.44	7874.34	17.89	0.57	18.11	0.57	18.33	0.67	19.11	0.88	19.44	0.96
32	1	25688.00	0.00	17.00	0.00	18.00	0.00	18.00	0.00	18.00	0.00	19.00	0.00

Table A.2: The last iterations and post replication values for $z_s = 100$ and $\beta = 10.4$

Last	Sims	Replication		93%		94%		95%		96%		97%	
Iter		Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
18	8	8107.38	1919.16	12.88	0.33	13.00	0.00	13.13	0.33	13.63	0.48	14.00	0.00
19	122	10764.67	2638.72	13.02	0.22	13.19	0.41	13.59	0.49	13.95	0.25	14.16	0.39
20	361	11676.13	2736.49	13.10	0.31	13.38	0.49	13.78	0.44	14.04	0.29	14.42	0.51
21	297	12517.26	2796.20	13.17	0.38	13.49	0.51	13.83	0.44	14.13	0.38	14.57	0.53
22	159	12831.63	2598.74	13.22	0.44	13.59	0.53	13.88	0.44	14.21	0.44	14.70	0.55
23	41	13072.93	3149.25	13.22	0.41	13.44	0.50	13.98	0.35	14.12	0.33	14.59	0.49
24	9	13220.44	2763.93	13.33	0.47	13.56	0.50	13.89	0.31	14.33	0.67	14.89	0.31
25	2	9826.50	1950.50	13.00	0.00	14.00	0.00	14.00	0.00	14.00	0.00	14.50	0.50
26	1	7936.00	0.00	14.00	0.00	14.00	0.00	14.00	0.00	15.00	0.00	16.00	0.00

Appendix B

List of symbols used for PAC'nPost

The symbols used for PAC'nPost are:

- n : The number of nodes in the network.
- s : The number of iterations between the scheduled requests, i.e. a node makes its scheduled request every s iterations.
- r : The number of nodes a blog post is replicated onto.
- z : The number of nodes queried to retrieve a blog post.
- m : The number of unique documents in a PAC network.
- ρ : The capacity of each node, i.e. the number of unique documents it can store.
- $\mathcal{T}$: The transfer buffer.
- μ : The PAC accuracy.
- s_r : The number of iterations a blog post is given to replicate in the number before it is made available for retrieval.
- β : The dissemination parameter - this controls the probability of a blog post being selected into a node's buffer.
- z_s : The number of peers contacted by a node when it publishes a post. This is the post seeding. parameter.
- α : The blog post creation rate per iteration.
- f : The number of followers a node has within the network.
- c_t : The cost in bytes, when one node contacts another. This is the size of an empty TCP/IP packet, and the cost must be incurred in all requests and responses.
- c_n : The cost in bytes of specifying a peer in a request.
- c_b : The size, in bytes of a micro-blog post.

- z_h : The number of nodes queried to retrieve a post published by a high popularity node.
- z_g : The number of nodes queried to retrieve a post published by a regular popularity node.
- r_h : The number of nodes a post published by a high popularity node is replicated onto.
- r_g : The number of nodes a post published by a regular popularity node is replicated onto.
- δ : The parameter for selecting a node to act as a replicator.
- $\mathcal{R}$: The set of nodes acting as *replicators* at an iteration.
- z_r : The number of nodes contacted by a *replicator*.
- y : The fraction of posts in which the trending keyword appears in.
- j : The number of keyword-frequency pairs returned by a node.
- w : System parameter for selecting high frequency keywords.
- σ : The trending fraction.
- $\mathbf{Bw}$: The bandwidth for trending keywords.

Appendix C

Publications

1. H. Asthana, Ruoxun Fu, and Ingemar J. Cox. On the feasibility of unstructured peer-to-peer information retrieval. In *Proceedings of the Advances in Information Retrieval Theory - Third International Conference, ICTIR 2011, Bertinoro, Italy*, 2011.

2. H. Asthana and Ingemar J. Cox. Pac'npost: a framework for a micro-blogging social network in an unstructured p2p network. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, 2012.

3. H. Asthana and Ingemar J. Cox. Retrieval of highly dynamic information in an unstructured peer-to-peer network. In *Proceedings of the 10th workshop on Large-scale and distributed informational retrieval*, 2013.[1]

4. H. Asthana and Ingemar J. Cox. A framework for peer-to-peer micro-blogging. In *Proceedings of the 33rd International Conference on Distributed Computing SystemsWorkshops (ICDCSW)*, 2013.

5. H. Asthana and Ingemar J. Cox. Retrieval of trending keywords in a peer-to-peer microblogging OSN. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management, CIKM13, San Francisco, CA, USA*, 2013.

6. H. Asthana and Ingemar J. Cox. Trending topics in a peer-to-peer micro-blogging social network. In *Proceedings of the 13th International Conference on Peer-to-Peer Computing (P2P), IEEE*, 2013.

[1] Best paper award

Bibliography

[AC12] H. Asthana and Ingemar J. Cox. Pac'npost: a framework for a micro-blogging social network in an unstructured p2p network. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, 2012.

[AC13a] H. Asthana and Ingemar J. Cox. A framework for peer-to-peer micro-blogging. In *Proceedings of the 33rd International Conference on Distributed Computing Systems Workshops (ICDCSW), IEEE*, 2013.

[AC13b] H. Asthana and Ingemar J. Cox. Retrieval of highly dynamic information in an unstructured peer-to-peer network. In *Proceedings of the 10th workshop on Large-scale and distributed informational retrieval*, 2013.

[AC13c] H. Asthana and Ingemar J. Cox. Retrieval of trending keywords in a peer-to-peer micro-blogging osn. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, 2013.

[AC13d] H. Asthana and Ingemar J. Cox. Trending topics in a peer-to-peer micro-blogging social network. In *Proceedings of the 13th International Conference on Peer-to-Peer Computing (P2P), IEEE*, 2013.

[ACD$^+$09] A. Apolloni, K. Channakeshava, L. Durbeck, M. Khan, C. Kuhlman, B. Lewis, and S. Swarup. A study of information diffusion over a realistic social network model. In *2009 International Conference on Computational Science and Engineering*, pages 675–682. IEEE, 2009.

[AFC11] H. Asthana, Ruoxun Fu, and Ingemar J. Cox. On the feasibility of unstructured peer-to-peer information retrieval. In *Proceedings of the Advances in Information Retrieval Theory - Third International Conference, ICTIR 2011, Bertinoro, Italy, September 12-14, 2011*, 2011.

[ALPH01] Lada A Adamic, Rajan M Lukose, Amit R Puniyani, and Bernardo A Huberman. Search in power-law networks. *Physical review E*, 64(4):046135, 2001.

[Bai75] N.T.J. Bailey. *The mathematical theory of infectious diseases and its applications*. Charles Griffin & Company Ltd, 5a Crendon Street, High Wycombe, Bucks HP13 6LE., 1975.

[BBC11] BBC. South tyneside council gets twitter data in blog case. `http://www.bbc.co.uk/news/uk-england-tyne-13588284`, 2011.

[BBC12a] BBC. Cat and mouse game with china's censors. `http://www.bbc.co.uk/news/world-asia-china-20293514`, 2012.

[BBC12b] BBC. Twitter to selectively censor tweets by country. `http://www.bbc.co.uk/news/world-us-canada-16753729`, 2012.

[BBC13] BBC. China employs two million microblog monitors state media say. `http://www.bbc.co.uk/news/world-asia-china-24396957`, 2013.

[BGK+09] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009.

[BMT+05] Matthias Bender, Sebastian Michel, Peter Triantafillou, Gerhard Weikum, and Christian Zimmer. Minerva: Collaborative p2p search. In *Proceedings of the 31st international conference on Very large data bases*, pages 1263–1266. VLDB Endowment, 2005.

[BOS12] David Bamman, Brendan O'Connor, and Noah Smith. Censorship and deletion practices in chinese social media. *First Monday*, 17(3), 2012.

[BSVD09] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2p social networking: early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52. ACM, 2009.

[BYGJ+07] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, and F Silvestri. The impact of caching on search engines. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, July 23-27, 2007, Amsterdam, The Netherlands*. ACM, 2007.

[BYL09] John Buford, Heather Yu, and Eng Keong Lua. *P2P networking and applications*. Morgan Kaufmann, 2009.

[CFH09] Ingemar J Cox, Ruoxun Fu, and Lars Kai Hansen. Probably approximately correct search. In *Advances in Information Retrieval Theory*, pages 2–16. Springer, 2009.

[CS02] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2002.

[CZFH10] Ingemar Cox, Jianhan Zhu, Ruoxun Fu, and Lars Kai Hansen. Improving query correctness using centralized probably approximately correct (pac) search. In *Advances in Information Retrieval*, pages 265–280. Springer, 2010.

[DGH^{+}87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, 1987.

[Dom05] P. Domingos. Mining social networks for viral marketing. *IEEE Intelligent Systems*, 2005.

[EGKM04] P.T. Eugster, R. Guerraoui, A.M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, 2004.

[FRA^{+}05] R.A. Ferreira, M.K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan. Search with probabilistic guarantees in unstructured peer-to-peer networks. In *P2P*, 2005.

[GD05] Saikat Guha and Neil Daswani. An experimental study of the skype peer-to-peer voip system. Technical report, Cornell University, 2005.

[Goo12] Google. Building a better shopping experience. `http://googlecommerce.blogspot.co.uk/2012/05/building-better-shopping-experience.html`, 2012.

[Har91] Donna Harman. How effective is suffixing? *JASIS*, 42(1):7–15, 1991.

[Het00] Herbert W Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.

[JMB05] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems (TOCS)*, 23(3):219–252, 2005.

[JSFT07] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 56–65. ACM, 2007.

[JVG^{+}07] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)*, 25(3):8, 2007.

[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 482–491. IEEE, 2003.

[KLPM10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[KSG13] Michal Kosinski, David Stillwell, and Thore Graepel. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences*, 110(15):5802–5805, 2013.

[LAH07] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007.

[LCC$^+$02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th international conference on Supercomputing*, pages 84–95. ACM, 2002.

[LIJM$^+$10] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 75–86. ACM, 2010.

[LLH$^+$03] J. Li, B. T. Loo, J Hellerstein, F. Kaashoek, and D. R. Karger. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03), Berkeley, CA, USA*, pages 207–215, 2003.

[LNTM11] Andreas Loupasakis, Nikos Ntarmos, Peter Triantafillou, and Darko Makreshanski. exo: Decentralized autonomous scalable social networking. In *CIDR*, pages 85–95, 2011.

[LPR07] Joao Leitao, Jose Pereira, and Luis Rodrigues. Epidemic broadcast trees. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 301–310. IEEE, 2007.

[Min09] Minnesota. Minnesota internet traffic studies (mints). `http://dtc.umn.edu/mints/`, 2009.

[MM02] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[MNP04] Yamir Moreno, Maziar Nekovee, and Amalio F Pacheco. Dynamics of rumor spreading in complex networks. *Physical Review E*, 69(6):066130, 2004.

[MRS08] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[MRW97] D. Malkhi, M. Reiter, and R. Wright. Probabilistic quorum systems. In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, 1997.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.

[New02] Mark EJ Newman. Spread of epidemic disease on networks. *Physical review E*, 66(1):016128, 2002.

[NMBM07] Maziar Nekovee, Yamir Moreno, G Bianconi, and M Marsili. Theory of rumour spreading in complex social networks. *Physica A: Statistical Mechanics and its Applications*, 374(1):457–470, 2007.

[NPA11] Rammohan Narendula, Thanasis G Papaioannou, and Karl Aberer. My3: A highly-available p2p-based online social network. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pages 166–167. IEEE, 2011.

[OMLS11] Iadh Ounis, Craig Macdonald, Jimmy Lin, and Ian Soboroff. Overview of the trec-2011 microblog track. In *Proceeddings of the 20th Text REtrieval Conference (TREC 2011)*, 2011.

[PGES05] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV*, pages 205–216. Springer, 2005.

[PGW$^+$08] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, and D.H.J. Epema. Tribler: a social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 2008.

[Pro13] Propublica. Chinas memory hole: The images erased from sina weibo. `https://projects.propublica.org/weibo/`, 2013.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[RFH$^+$01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.

[RJ76] S.E. Robertson and K.S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

[RWJ$^+$96] S. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *Proc. of the Third Text REtrieval Conference (TREC 1994)*, pages 109–126, 1996.

[San11] Sandvine. Sandvine global internet phenomena report: Spring 2011. `http://www.sandvine.com/downloads/documents/05-17-2011\_phenomena/Sandvine\%20Global\%20Internet\%20Phenomena\%20Report\%20-\%20ALL.zip`, 2011.

[SM02] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *Peer-to-Peer Systems*, pages 261–269. Springer, 2002.

[SMK$^+$01] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 149–160. ACM, 2001.

[SR06] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[SYBA10] Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon. *Handbook of peer-to-peer networking*, volume 1. Springer, 2010.

[Sys09] Sysomos. Inside twitter. `http://www.sysomos.com/insidetwitter/`, 2009.

[Tec12] Techcrunch. Twitter passes 200m monthly active users. `http://techcrunch.com/2012/12/18/twitter-passes-200m-monthly-active-users-a-42-increase-over-9-months/`, 2012.

[Tel13] Telegraph. Twitter in numbers. `http://www.telegraph.co.uk/technology/twitter/9945505/Twitter-in-numbers.html`, 2013.

[TKLB07a] Wesley W Terpstra, Jussi Kangasharju, Christof Leng, and Alejandro P Buchmann. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 49–60. ACM, 2007.

[TKLB07b] Wesley W Terpstra, Jussi Kangasharju, Christof Leng, and Alejandro P Buchmann. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 49–60. ACM, 2007.

[UPS11] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. A survey of dht security techniques. *ACM Computing Surveys (CSUR)*, 43(2):8, 2011.

[VGVS05] Spyros Voulgaris, Daniela Gavidia, and Maarten Van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[VVS03] S. Voulgaris and M. Van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. *Self-Managing Distributed Systems*, pages 299–308, 2003.

[XCFH10] Tianyin Xu, Yang Chen, Xiaoming Fu, and Pan Hui. Twittering by cuckoo: decentralized and socio-aware online microblogging services. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 473–474. ACM, 2010.

[YGM02] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 5–14. IEEE, 2002.

[YJC07] W-PK Yiu, Xing Jin, and S-HG Chan. Vmesh: Distributed segment storage for peer-to-peer interactive video streaming. *Selected Areas in Communications, IEEE Journal on*, 25(9):1717–1731, 2007.