QLectives – Socially Intelligent Systems for Quality
Project no. 231200

Instrument: Large-scale integrating project (IP)
Programme: FP7-ICT

Deliverable D4.3.4
QMedia v4 - Short report

Submission date: 2013-02-17

Start date of project: 2009-03-01 Original duration: 48 months

Organisation name of lead contractor for this deliverable: TUDelft

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidental, only for members of the consortium (including the Commission Services)	

Document information

1.1 Author(s)

Author	Organisation	E-mail
Niels Zeilemaker	TU Delft	n.zeilemaker@tudelft.nl
Boudewijn Schoon	TU Delft	p.b.schoon@tudelft.nl
Johan Pouwelse	TU Delft	J.A.Pouwelse@tudelft.nl

1.2 Other contributors

Name	Organisation	E-mail

1.3 Document history

Version#	Date	Change
V0.1	December 1, 2012	Starting version, template
V0.5	December 13, 2012	First draft
V0.9	December 16, 2012	Complete first draft, for internal consortium
V1.0	17 February, 2013	Approved version to be submitted to EC

1.4 Document data

Keywords	peer-to-peer, distributed mass media, decentralized channels, decentralized wiki
Editor address data	J.A.Pouwelse@tudelft.nl
Delivery date	17 February, 2013

1.5 Distribution list

Date	Issue	E-mail
	Consortium members	QLECTIVES@list.surrey.ac.uk
	Project officer	Roumen.BORISSOV@ec.europa.eu
	EC archive	INFSO-ICT-231200@ec.europa.eu

QLectives Consortium

This document is part of a research project funded by the ICT Programme of the Commission of the European Communities as grant number ICT-2009-231200.

University of Surrey (Coordinator)
Department of Sociology/Centre
for Research in Social Simulation
Guildford GU2 7XH
Surrey
United Kingdom
Contact person: Prof. Nigel Gilbert
E-mail: n.gilbert@surrey.ac.uk

University of Fribourg
Department of Physics
Fribourg 1700
Switzerland
Contact person: Prof. Yi-Cheng Zhang
E-mail: yi-cheng.zhang@unifr.ch

Technical University of Delft
Department of Software Technology
Delft, 2628 CN
Netherlands
Contact Person: Dr Johan Pouwelse
E-mail: j.a.pouwelse@tudelft.nl

University of Warsaw
Faculty of Psychology
Warsaw 00927
Poland
Contact Person: Prof. Andrzej Nowak
E-mail: nowak@fau.edu

ETH Zurich
Chair of Sociology, in particular
Modelling and Simulation
Zurich, CH-8092
Switzerland
Contact person: Prof. Dirk Helbing
E-mail: dhelbing@ethz.ch

Centre National de la Recherche Scientifique, CNRS
Paris 75006,
France
Contact person: Dr. Camille ROTH
E-mail: camille.roth@polytechnique.edu

University of Szeged
MTA-SZTE Research Group on
Artificial Intelligence
Szeged 6720, Hungary
Contact person: Dr Mark Jelasity
E-mail: jelasity@inf.u-szeged.hu

Institut für Rundfunktechnik GmbH
Munich 80939
Germany
Contact person: Dr. Christoph Dosch
E-mail: dosch@irt.de

QLectives introduction

QLectives is a project bringing together top social modelers, peer-to-peer engineers and physicists to design and deploy next generation self-organising socially intelligent information systems. The project aims to combine three recent trends within information systems:

- **Social networks** - in which people link to others over the Internet to gain value and facilitate collaboration
- **Peer production** - in which people collectively produce informational products and experiences without traditional hierarchies or market incentives
- **Peer-to-Peer systems** - in which software clients running on user machines distribute media and other information without a central server or administrative control

QLectives aims to bring these together to form Quality Collectives, i.e. functional decentralised communities that self-organise and self-maintain for the benefit of the people who comprise them. We aim to generate theory at the social level, design algorithms and deploy prototypes targeted towards two application domains:

- **QMedia** - an interactive peer-to-peer media distribution system (including live streaming), providing fully distributed social filtering and recommendation for quality
- **QScience** - a distributed platform for scientists allowing them to locate or form new communities and quality reviewing mechanisms, which are transparent and promote

The approach of the QLectives project is unique in that it brings together a highly inter-disciplinary team applied to specific real world problems. The project applies a scientific approach to research by formulating theories, applying them to real systems and then performing detailed measurements of system and user behaviour to validate or modify our theories if necessary. The two applications will be based on two existing user communities comprising several thousand people - so-called ”Living labs”, media sharing community tribler.org; and the scientific collaboration forum EconoPhysics.

Executive summary

This report accompanies and documents *software* deliverable QMedia version 4.0. This is thus a technical manual for an Internet-deployed self-organising system.

This is an *incremental* deliverable, contents of prior QMedia deliverables is only repeated when needed for readability. QMedia is a next-generation social media distribution platform and one of QLectives living labs. Its vision is to provide an alternative to mass media through the concept of self-organising personalised collectives. Version 4.0 of the QMedia software is based on the QLectives Platform version 4.0 (D.4.1.4).

The main contributions of this deliverable are:

- *A smartphone version of QMedia* in line with the "mobile first" strategy emerging globally we succeeded in both moving self-organising algorithms to energy and computationally limited devices plus utilising the novel QPlatform no-Internet-needed option (a world-first innovation);
- *Automatic quality testing ecosystem* created daily report on the quality and performance of QMedia software;
- *Promote cooperation using User Interaction Strength* we experimentally evaluate using effort to provide preferential treatment;
- *Community building* analysis of how to continue QMedia in the long term;

QMedia version 4.0 has been released in early December 2012. This version of the software, named Tribler 6.0.4 was downloaded more than 25,000 times since its publication[1].

[1] for more statistics, see `http://statistics.tribler.org`

Contents

Chapter 1

Introduction

This report describes QMedia version 4.0, the fourth version of an experimental media-sharing distributed software built around the concept of quality collectives. The network of QMedia users serves as a living lab for QLectives, both as a testbed for developed algorithms and providing input for the design and evaluation of new algorithms.

QMedia's vision is to provide a decentralized, user-centric and social media-distribution platform. This platform can provide an alternative to mass media through self-organising personalised collectives.

QMedia is based on the QLectives Platform (see D4.1.1, D4.1.2 and D4.1.3 for details on the platform), which aims to serve as a generic middleware to develop peer-to-peer (P2P) applications using its central component, a Distributed Permission System, called Dispersy (reported in D4.1.4). The recent development in the QLectives Platform is reported in D4.1.4, here we are focusing on the recent development in QMedia.

Chapter 2

Smartphone support for QMedia

On December 2012 we released a smartphone version of QMedia.

Signifcant resources where spend to move part of the QMedia functionality to the Android OS. Under the Tribler brandname we released this code publically and issued a local press release. This activity was a joined effort by both EIT funded researchers and QLectives scientists. Unfortunately, we lack the resources to create a mature smartphone app for a large audience. Significant effort is required to refine our operational app into a smooth and polished version.

The local press release announcing our app is included below in full.

Tribler Mobile: share videos, even without the internet

With the advent of smartphones, sharing videos has become increasingly popular. To do it, users are also increasingly dependent on fast and expensive mobile data connections and a limited number of well-known video websites. This is why researchers at TU Delft have joined forces with their counterparts at Swedish University KTH and the French institute INRIA to design a mobile app that makes it possible to share and distribute videos, even without an internet connection: Tribler Mobile.

The app makes use of the wireless connections between phones and peer-to-peer (p2p) technology. The beta-version of Tribler Mobile, for Android phones with Near Field Communication (NFC)- is now available as a free download at the Android Market.

Cloud

Researcher Johan Pouwelse from TU Delft is the brains behind the p2p program Tribler. The app we are developing enables people to distribute videos by

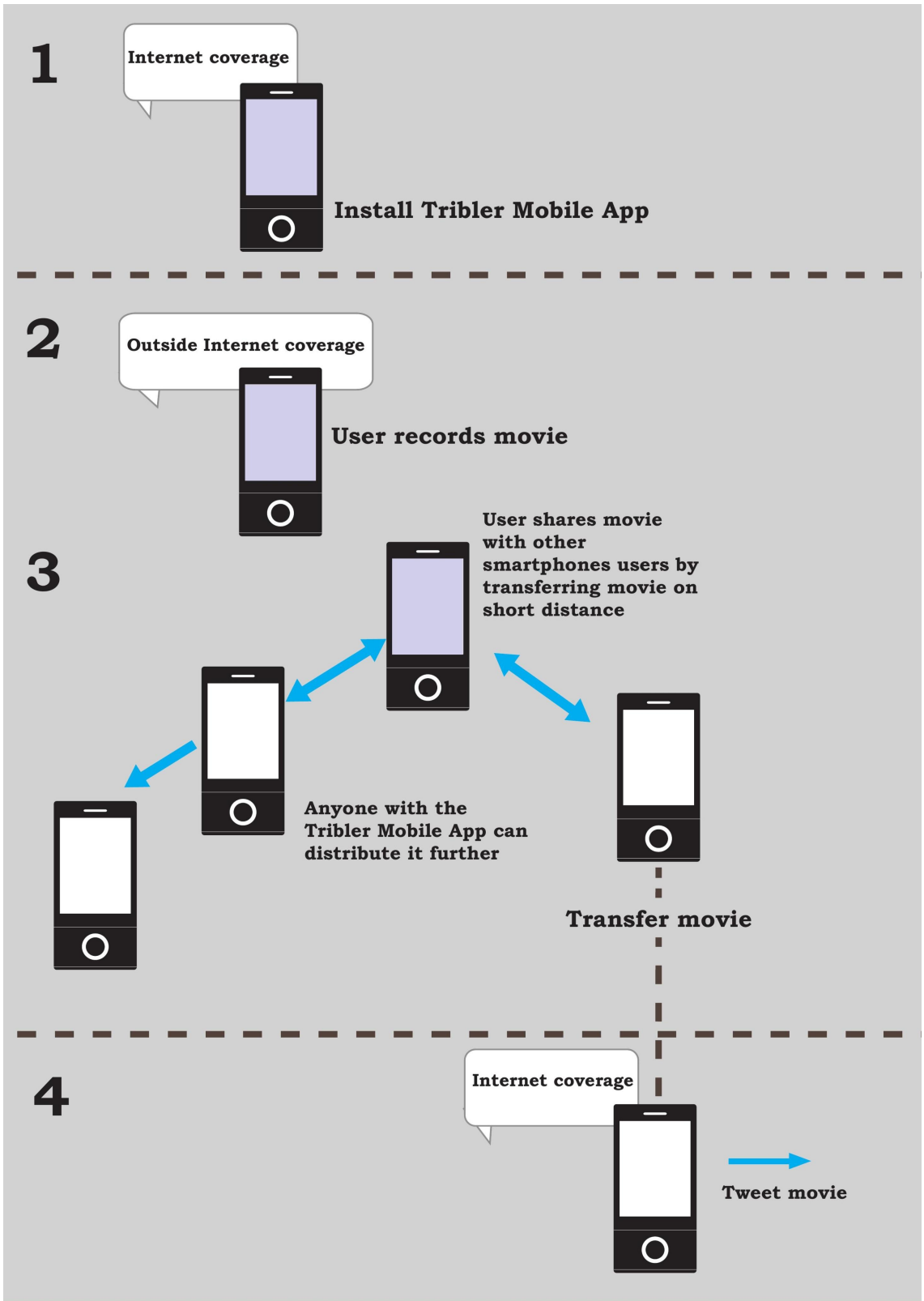


Figure 2.1: Infographics of launched smartphone app.

copying them from phone to phone wirelessly. Anyone who then has an internet connection that works can distribute the video using cloud technology. People will no longer be reliant on video websites like YouTube.

Live broadcast

Many smartphones have data limits and these deter people from uploading video files. Now, you can share the video with friends simply by holding your phones against each other. You can also use it abroad, where data usage can be expensive, or if there is no mobile coverage.

Another feature that Tribler Mobile offers is that the app enables users with an internet connection to broadcast live from their mobile. People watching the live broadcast are not only viewing it themselves; their mobile or laptop internet connection is also helping to distribute the broadcast to many more computers and phones, claims Pouwelse. It is much cheaper and less vulnerable to overloading than a web stream from a server.

New standard

Pouwelse's group is the world's largest research group that specialises in peer-to-peer systems. The app is open source and is based on a new internet standard called PPSP that makes high-quality video possible for smartphones, tablets and internet-enabled TVs, without high energy consumption. The standard was designed by Pouwelse's team. The researchers are collaborating with such organisations as Wikipedia and the BBC, which have both conducted extensive public internet tests in order to establish exactly how much money is saved. The project is funded by the European Union 7th framework research program and the European Institute of Innovation and Technology.

Chapter 3

Automated performance and quality testing

To improve the quality of our software we created a quality improvement environment. Every day an automatic code quality report is generated.

To enhance quality, Dispersy has been integrated in Jenkins: the daily testing framework. Dozens of tests have been defined to measure and quantify the fitness of our technology. We monitor multiple aspect such as CPU-, bandwidth-, disk-, and memory usage. A number of specific experiments quantify the pure data dissemination efficiency.

A performance test typically runs for 15 minutes. During its run time we monitor the different performance indicators: CPU, bandwidth, disk, and memory usage. We define a baseline for each of these performance indicators, i.e. bandwidth usage should never exceed 10MB of data. When a daily performance test exceeds one of the baseline requirements, a notification is given. From here on it is possible to determine the change, that must have taken place within the last 24 hours, that caused the performance to degrade. The aim is to consistently improve quality and never to degrade it.

3.1 Dissemination performance

Our data dissemination performance and quality test is designed to transfer a large amount of data between peers. Given that the conditions that the test runs under are the same for every run, it is reasonable to assume that the same amount

of CPU, bandwidth, disk, and memory should be consumed. Hence, any changes to the software that negatively influence the resource consumption can be detected.

The data dissemination test follows the following steps:

1. all M peers start, evenly spread across multiple DAS2 computing nodes;
2. a subsection M_{full} of the peers start with a filled database containing N data entries, the other M_{empty} peers start with an empty database;
3. all peers join the same community;
4. all peers perform the Dispersy semi-random walk, building the overlay and disseminating the N data entries;
5. all peers stop 15 minutes after the test started;
6. all logs are parsed and graphs are generated;
7. performance is analyzed and the test is marked as either success or fail.

To further facilitate performance testing, it is possible to run multiple dissemination tests under varying conditions, such as:

- M: the total number of peers in the community;
- M_{full}: the number of peers who start with a full database;
- $M_{empty} = M - M_{full}$: the number of peers who start with an empty database;
- N: the number of data entries in the full database;
- enable_sync_cache: with or without bloom filter caching;
- enable_sync_skip: with or without bloom filter skipping;
- sync_response_limit: larger or smaller bloom filter response bandwidth limits.

Where each set of conditions has its own upper limits assigned to CPU, bandwidth, disk, and, memory consumption. The following resource usage figures 3.1-3.1 result from a test containing $M = 500$ peers, where $M_{full} = 250$ peers have

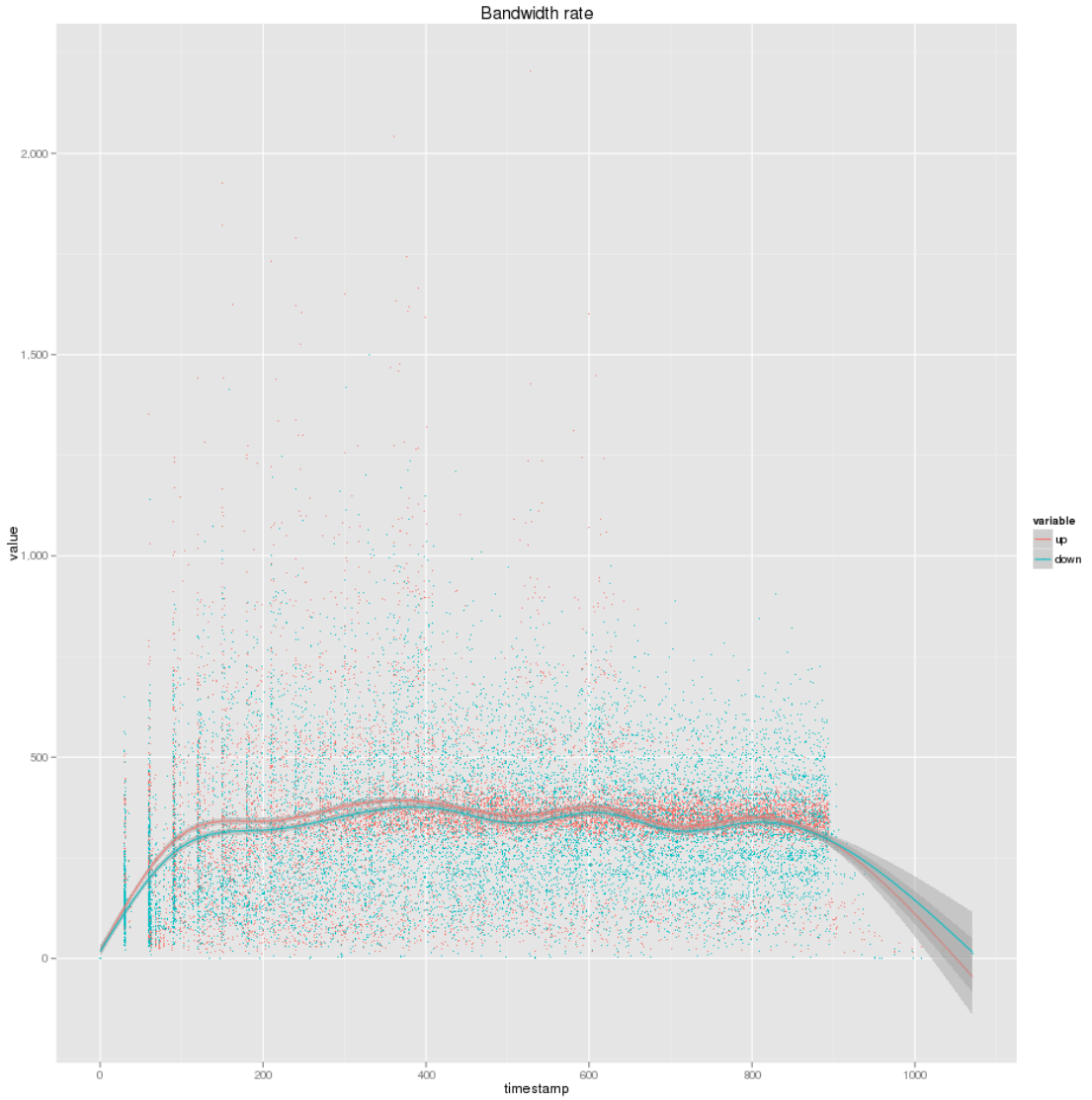


Figure 3.1: Baseline bandwidth usage

$N = 100,000$ data entries, with bloom filter caching and skipping disabled, and finally, where the bandwidth limit is 5120 bytes per received bloom filter. These figures will form our baseline that we will use to for comparison after two performance improvements.

Figure 3.1 shows the average up- and download bandwidth in bytes per second for all M peers. Clearly there are individual peers with bandwidth spikes reaching as high as 2000 B/s. This can occur when one peer received multiple sync bloom filters in close proximity, resulting in multiple responses.

Figure 3.1 shows the average CPU usage on each of the DAS nodes. The ini-

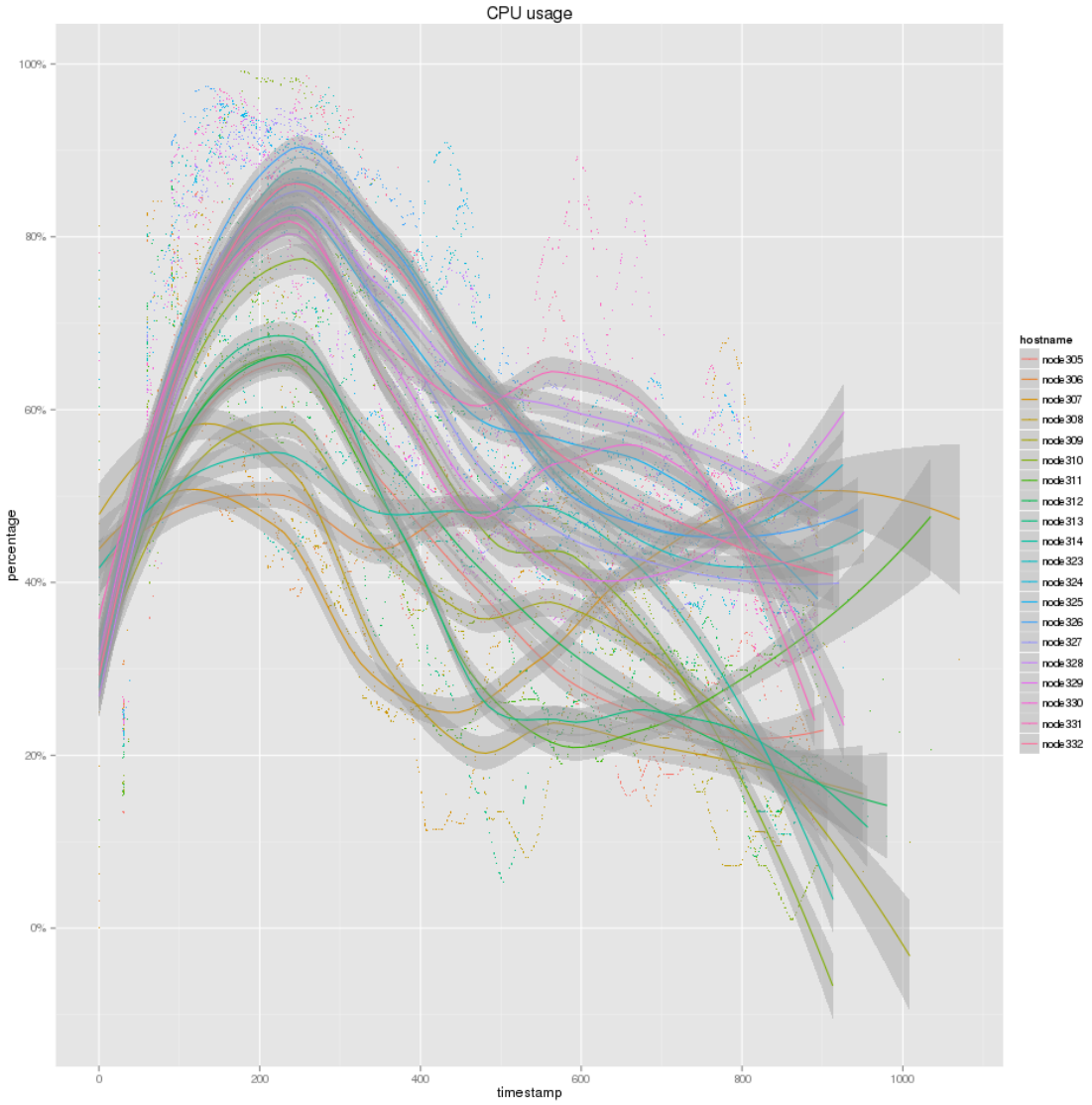


Figure 3.2: Baseline CPU usage

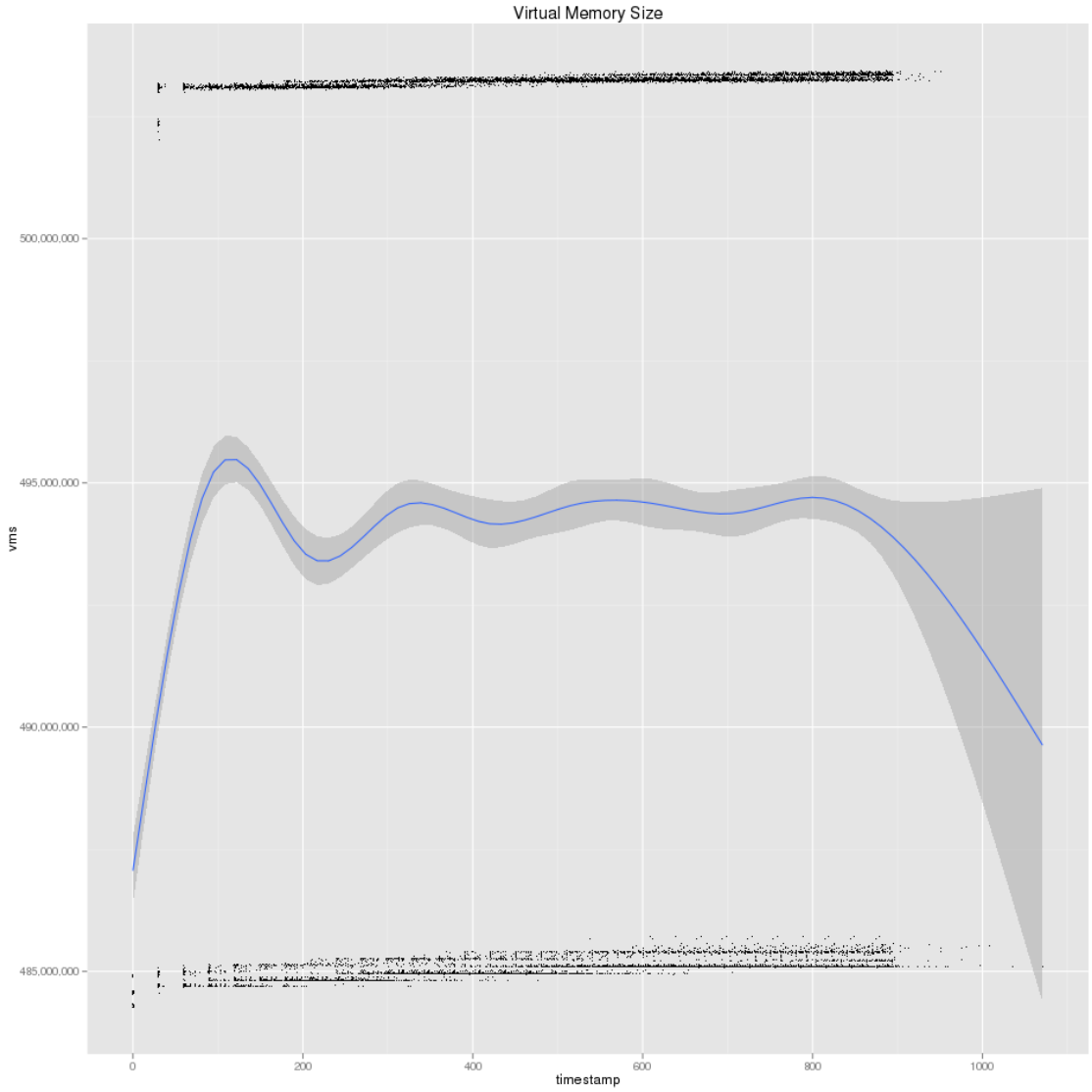


Figure 3.3: Baseline memory usage

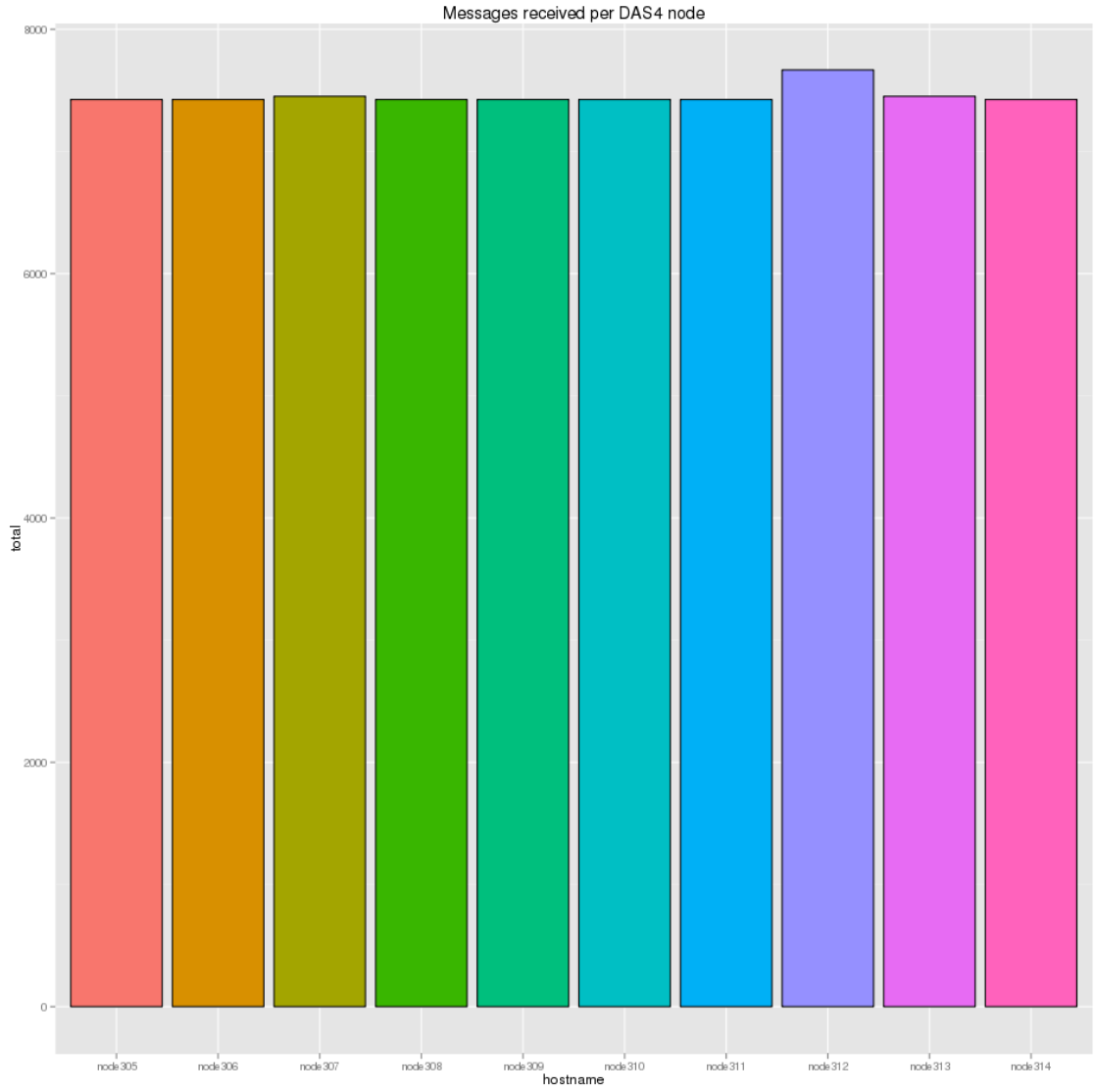


Figure 3.4: Baseline data entries received

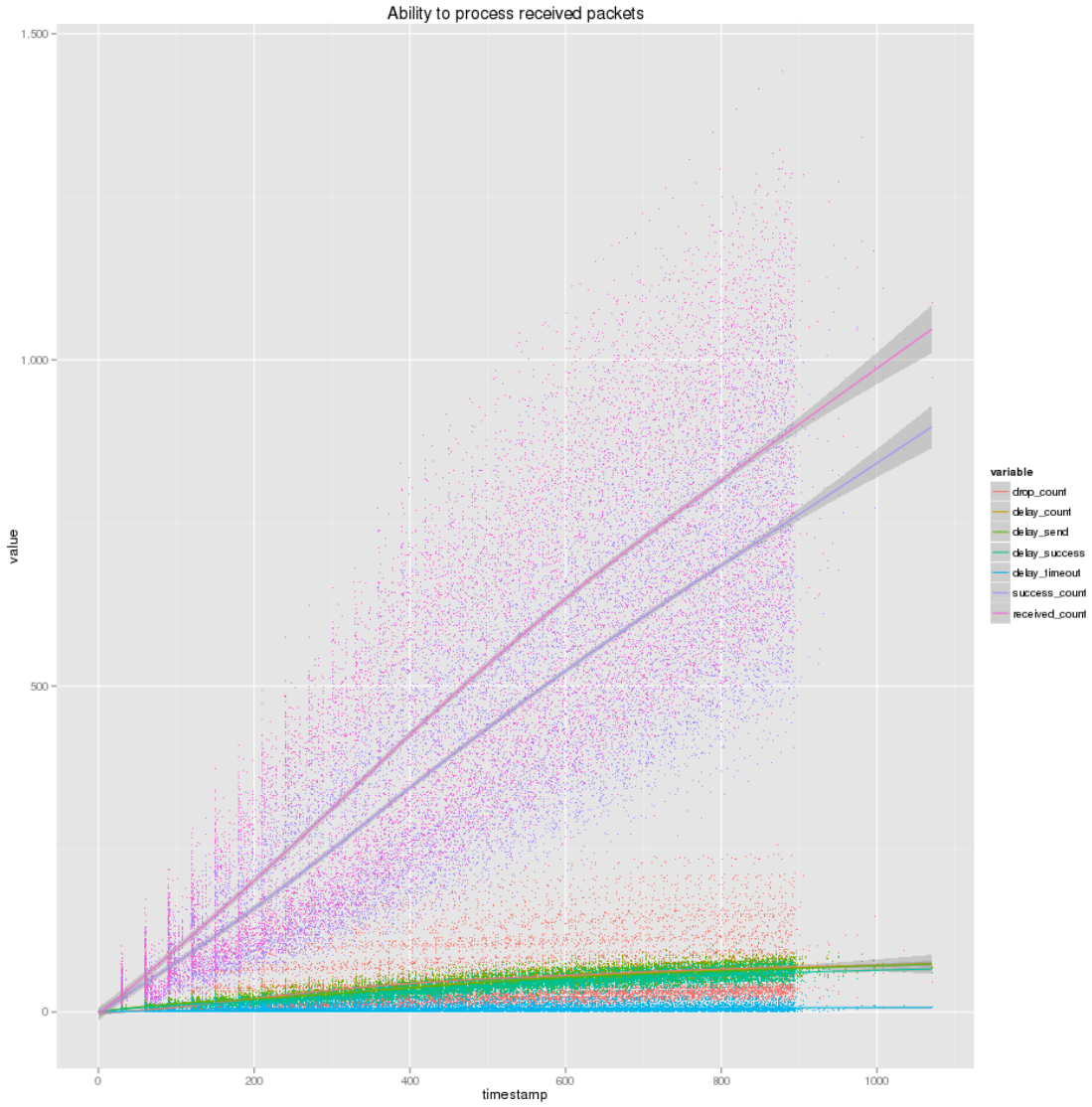


Figure 3.5: Baseline UDP packets received, delayed, and dropped

tial high CPU load is caused by starting all the individual python processes and loading the databases for the first time.

Figure 3.1 shows the average memory usage for all M peers. The figure shows that approximately half the measurements are in the top half and the other measurements are in the bottom half. These are measurements from the M_{full} and M_{empty} peers, respectively. From this figure we can see that memory usage remains constant throughout the experiment. Increasing memory usage would, most likely, indicate problems, such as a memory leak.

Figure 3.1 shows the number of data entries that were received per DAS node. Given that there were 25 peers running on each node, approximately $7500/25 = 300$ entries were received by each peer during the test.

Figure 3.1 shows various statistics. Most indicative are received_count and success_count. These indicate the number of UDP packets received and the number of packets successfully processed, respectively. A large difference between these values would indicate problems. The difference in this figure is expected.

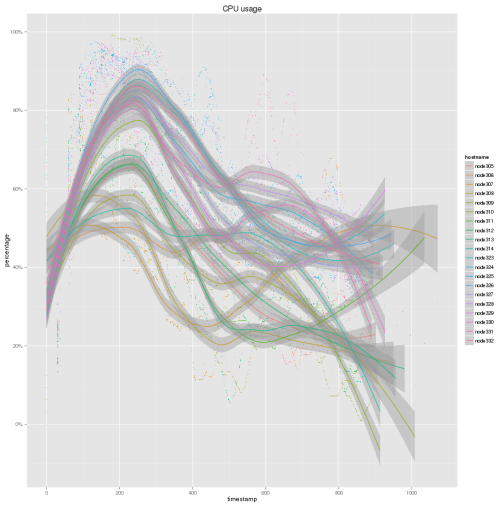
3.2 Performance improvement

Dispersy resource usage is, in large part, caused by the creation of the sync bloom filters. Hence, to limit resource usage, we should limit the creation of these bloom filters.

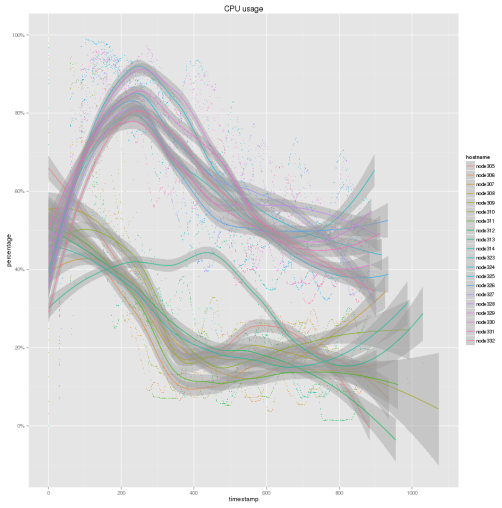
One strategy to accomplish this is to only create a new sync bloom filter whenever the current one no longer results in new data entries. This caching strategy is very effective at improving performance for peers that have recently joined the community, as they will be able to use the same bloom filter multiple times. The result of this strategy is shown in Figure 3.6(b). The figure clearly shows an improvement in the CPU usage of the nodes running the M_{empty} peers. The performance for the M_{full} peers remains unchanged, as expected.

Another strategy to create fewer bloom filters is to simply stop the synchronization process. Obviously this has the disadvantage of a peer not being able to obtain new data entries while synchronization is disabled, therefore we are currently investigating the proper parameters to obtain a good trade off between CPU and dissemination performance.

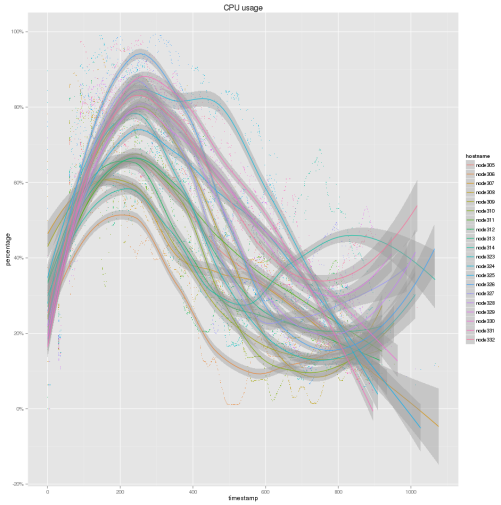
The current sync bloom filter skipping strategy will start skipping more sync



(a) Baseline CPU usage



(b) CPU usage with caching



(c) CPU usage with caching and skipping

Figure 3.6: CPU usage graphs

cycles as long as no new data entries are received from sending a bloom filter. The first time no new data entries are received only a single bloom filter is skipped, the second time two are skipped, etc. This internal counter is reset whenever new data entries are received. Furthermore, no more than 6 skips are allowed before a new bloom filter is made, hence each peer will create a bloom filter at least every $5 * 6 = 30$ seconds.

The result of this strategy is shown in Figure 3.6(c). As expected the performance for the M_{full} peers has improved. However, the performance for the M_{empty} peers has decreased.

This surprising result is explained by the total workload of the system. While the baseline experiment, see Figure 3.1, transferred approximately 7500 data entries per DAS node, the improvements gained from caching an skipping has increased the throughput to approximately 11,000 data entries. In other words, because the M_{full} peers are less busy building sync bloom filters, they have more resources available to more effectively disseminate data entries to the M_{empty} peers. This, in turn, results in higher CPU usage to process the additional data entries.

Hence, our performance tests have resulted in faster code and better algorithms causing the dissemination speed to increase. Furthermore, the nightly builds will ensure that the quality and performance remains at our standards.

Chapter 4

Promote cooperation using User Interaction Strength

This chapter consists of unpublished work submitted to the top venue in networking research: ACM SIGCOMM.

Online networks like Facebook and BitTorrent are based on user interactions such as wall posts and content exchange. In such systems, user relationships can be used to enhance security and promote cooperation, but in order to be meaningful, these relationships should be based on user interaction strength instead of "binary" friendships. To date, several theoretical, centralized schemes for *estimating user interaction strength* have been proposed. Here we present the design, deployment, and analysis of the UISE scheme for User Interaction Strength Estimation for both centralized and decentralized online networks.

Among the strong points of UISE is that it captures direct and indirect user interactions, that it scales with only partial information dissemination in decentralized systems, and that it provides disincentives for malicious user behaviors. We apply UISE to detect user interaction patterns based on wall posts in Facebook, which resemble those observed in the offline human society. We further apply UISE to online time estimation based on rendezvous as user interactions in Tribler, an online network for media and social applications like file sharing, streaming, and voting. We demonstrate the accuracy and scalability of UISE with different information dissemination protocols and user behaviors using simulations, emulations, and a real-world deployment.

4.1 Introduction

Online networks are complex distributed computer systems that involve potentially large numbers of humans with their inputs and decisions. Typical examples of online networks include email, Facebook, LinkedIn, Wikipedia, eBay, and BitTorrent-like Peer-to-Peer (P2P) systems. They have become popular and powerful infrastructures for communication and they provide various mechanisms for users to interact. For instance, in Facebook, users maintain friendships, post messages on their friends' walls, and comment on their friends' photos. In Wikipedia, users collectively edit articles in their areas of expertise, and in BitTorrent, users upload to and download from each other to share the contents of their common interests.

In this paper, we present a new view that sees *user interactions* as the most basic underpinning of online networks such as Facebook, Wikipedia, and BitTorrent. The key research question we address is whether we can devise a single primitive in such systems for expressing user interactions and their strengths that is both generic and can be applied to a wide range of systems.

The patterns and strengths of user interactions are prominent in online networks. For example, in BitTorrent, user interactions can be used as the foundation for designing incentive policies to promote contribution. Through estimating the interaction strengths between users in terms of the amounts or durations of uploads, system designers can make users favor the highly ranked users for future uploads. As another example, a number of applications [16, 17, 23, 24] leverage *online friendships* to enhance security, to promote cooperation, to improve item recommendation, etc. However, it has long been observed that low-interaction friendships, as exemplified by the "Familiar Stranger" [18], are prevalent, and it has been shown that the dynamics of user interactions is more representative for inferring user relationships [19, 21] than simple, statically established "binary" friendships. Distributed systems often rely on importing trust relationships from social networks such as Facebook to improve security [24]. Instead, users would be much better off by estimating their interaction strengths with others and by trusting the ones with whom they have interacted frequently.

The importance of user interactions in online networks leads to the question: *How can we estimate user interaction strength?* Previous work addressing this issue [2, 19, 21, 22] has focused only on online social networks like Facebook, and

has only considered *binary* user interactions, simply indicating whether a user has interacted with another user or not. To remedy this, in this paper we propose a User Interaction Strength Estimation scheme called UISE that has a much more fine-grained notion of user interaction and that is applicable to a more general category of online networks. With the astounding growth of online networks—with Facebook exceeding a billion users and BitTorrent serving hundreds of millions of users—UISE has a scalable design. Specifically, we make the following contributions.

As a model for representing user interaction histories, we introduce the *bitmap-based user interaction graph*, based on which UISE estimates user interaction strengths. UISE captures the frequencies of both direct and indirect interactions, provides disincentives for malicious user behaviors, and can be easily incorporated into distributed systems (Section 4.4).

We apply UISE to detect user interaction patterns in online networks. We take Facebook as an example and we derive patterns resembling those often observed in the offline human society—users in Facebook tend to interact frequently and stably with relatively small groups, occasionally with persons outside those groups, and they make new friends while in the meantime losing touch with some old friends (Section 4.5).

We further apply UISE to derive a decentralized scheme for estimating the time users are online, which is an important aspect of user activity and has yet not been studied before. We have implemented this application into the Tribler [13] online network and we demonstrate the scalability and the accuracy of UISE through simulations, emulations, and Internet deployment (Sections 4.6-4.9).

As it turns out, UISE achieves accurate estimations even when users hold only 5% of the global information related to user interactions. In fact, in order to maintain the accuracy of UISE, its requirement for the coverage of global information decreases with the population size, thus allowing UISE to achieve good scalability in a self-organized manner. Furthermore, although users only possess a partial view of the system, with UISE they can derive a ranking of users from their estimations that highly resembles the ranking derived from the global view—the most important application of estimating user interaction strength is to differentiate users with different levels of activity.

4.2 User interactions

Online networks provide various mechanisms for users to interact. In Facebook, users exchange messages, post on each other's walls, and comment on photos. In BitTorrent, where the main application is file sharing, users meet (rendezvous), and if possible, upload to and download from each other. As users evolve in online networks, they gradually build up a long history of user interactions. Some of these interactions are directed, such as uploads and downloads, while others are undirected, such as rendezvous and a chat over a photo. Online networks provide abundant user interactions, which, however, change rapidly over time: only 30% of Facebook users consistently interact from one month to the next [19], and in BitTorrent, users who directly download from each other for one file rarely meet again—the so-called *problem of low rendezvous*.

Fortunately, users not only interact directly, but also indirectly. An indirect interaction is formed when users are linked through a sequence of interactions, such as in Facebook when a user posts on another user's wall who in turn posts on a third user's wall, and so on, or in BitTorrent when a user uploads to another user who further uploads to a third user, and so forth. When direct interactions are relatively scarce, such as in BitTorrent-like P2P systems where the problem of low rendezvous exists, indirect interactions provide supplementary information for inferring user relationships. Moreover, a group of direct and indirect interactions that happen within a short time frame may be caused by offline interactions. In the Facebook example, the corresponding users could have participated in some offline event together and are sharing their experiences. Indirect interactions that happen widely apart in time, however, are of limited use. For example, a user may have exchanged messages with a high-school friend three years ago and with a college friend one hour ago—these interactions hardly indicate any meaningful user relationships between the user's two friends.

To achieve a meaningful estimation of user interaction strength, the above aspects of user interaction should be considered. In the following section, we give the problem statement and discuss the challenges that need to be addressed.

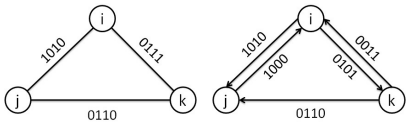
4.3 Problem statement

User interaction strength is reflected by two aspects: the frequency of interactions and the intensity of each interaction. In this paper, we do not consider the latter aspect in order to avoid evaluating the strengths of words, such as to decide which comment should get a higher weight, "Happy birthday" or "You look nice". Rather, in this paper we define user interaction strength as the *frequency that two users interact*, and we propose a model for estimating user interaction strength based on this definition. In this context, the following four issues are addressed.

Partial history versus full history of user interactions. A properly selected partial history of user interactions is more suitable for estimating user interaction strength than a full history. First, as user interactions may change rapidly over time, the stale ones are no longer useful for inferring meaningful relationships. Secondly, keeping a full history of user interactions induces a "glass ceiling" for new users: they will always be evaluated as less active than old users who have already accumulated a long history. When user interaction strengths are used to decide on the service level a user can get, such an unfair playing field will discourage new users from joining and will eventually decay the system.

Direct versus indirect interactions. As stated in Section 4.2, direct and indirect interactions are equally important and therefore should both be included. The only concern is that, as specified by the small world theory [10], indirect interactions might link all users together. A proper design should avoid this issue. In our model, we only consider indirect interactions that happened together with related direct interactions within a certain time period. For example, when a user exchanges messages with another user who further exchanges messages with a third user, the first and third users are only linked when these user interactions happened within say one week, with the length of the eligible period as a tunable parameter.

Scalability. As an online network evolves, users involve in a huge number of interactions. Somehow, user interaction records have to be collected and analyzed to estimate user interaction strengths, but doing so at central servers may be neither scalable nor practical. We propose a decentralized approach in which the collection of interaction records and the estimation of interaction strengths are performed by the end users, where the estimations are actually used. When



(a) Undirected (b) Directed

Figure 4.1: Examples of user interaction graphs

applied in such a distributed way, the design should be lightweight so that it will not impose a high computational load on the end users.

Security. Online networks are subject to security concerns. When user interaction strengths are used to decide on the service level a user can get, malicious users may disseminate false information to be better off than others, e.g., through Sybil attacks. Traditional defenses against these attacks rely on trusted identities provided by an authority or automatically imported from some other online social networks [9, 24]. However, requiring users to present trusted identities runs counter to the open membership that underlies the success of these systems in the first place, and inferring trustworthy relationships purely from online friendships has been proven to be insufficient [19, 21]. Without assuming secure super-nodes, we propose a design in which malicious users that disseminate false information will not gain any privileges, i.e., in which disincentives for malicious user behaviors are provided.

4.4 Design description

In this section, we introduce UISE, a user interaction strength estimation scheme that addresses the issues listed in Section 4.3. In UISE, users collect the whole or parts of the interaction histories of other user pairs through central servers or distributed information dissemination, based on which they estimate their user interaction strengths with other users. UISE consists of four parts: representing the user interaction history, estimating the user interaction strength, maintaining security against malicious user behaviors, and incorporating it into distributed systems. In the following sections, we discuss them in turn.

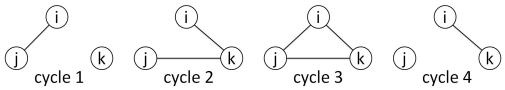


Figure 4.2: Per-cycle user interaction graphs generated from Fig. 4.1(a).

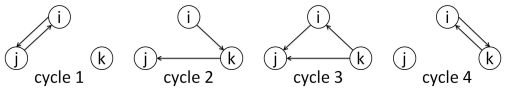


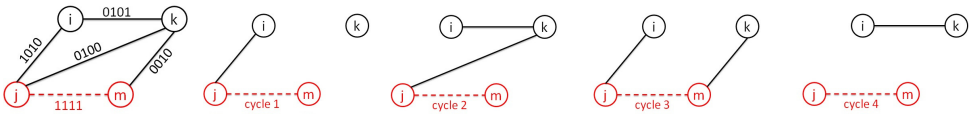
Figure 4.3: Per-cycle user interaction graphs generated from Fig. 4.1(b).

4.4.1 Representing user interaction history

In UISE, the interaction histories received by a user are incorporated into its *bitmap-based user interaction graph* (UIG), a model that we introduce to represent user relationships based on their interactions. Different users build different UIGs unless they can obtain full knowledge of the system, for example, through central servers. Therefore, a UIG reflects a user's local view of the system.

In a UIG, a vertex represents a user and the edge between two vertices represent the interaction history of the users it connects. The interaction history of two connected users is reflected by a label of the corresponding edge, which is a string called the *interaction bitmap*, or simply *bitmap*. To capture the interaction frequency, we abstract time into cycles where one cycle represents a certain unit of time such as 30 minutes. We keep the interaction history in a time-based (cycle-based) sliding window fashion, with the window size being equal to the length of the kept history. When two users have directly interacted in a particular cycle, the corresponding bit in their bitmap is set to 1 (otherwise it is set to 0). As time evolves, their interaction bitmap becomes a binary string and the number of "1"s shows how frequently they have recently interacted.

Fig. 4.1 shows examples of UIGs for undirected and directed user interactions, respectively, with a window size of 4. For example, when Fig. 4.1(a) is derived



(a) UIG (b) Per-cycle UIGs

Figure 4.4: An example of user interaction graph (UIG) with false information from user j and user m

from wall posts in Facebook, it specifies that users i and j have chatted on each other's wall in cycles 1 and 3. When the example is derived from the upload and download interactions in BitTorrent, Fig. 4.1(b) specifies that user i has uploaded to user k in cycles 2 and 4.

4.4.2 Estimating user interaction strength

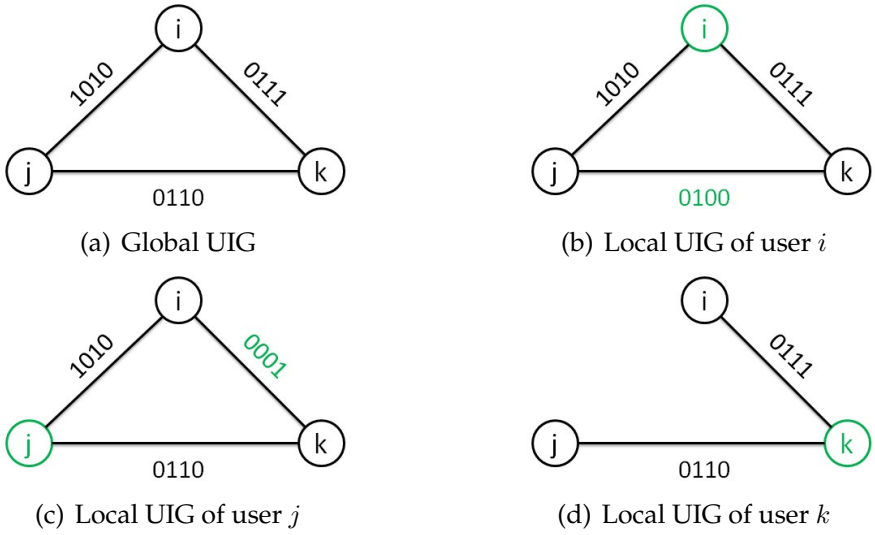
To calculate the interaction strength, i.e., the frequency that two users interact (directly or indirectly), we perform cycle by cycle examinations. First, a UIG is divided into a number of *per-cycle* UIGs. Then, for each of these per-cycle UIGs, an algorithm for finding *connected components* or *reachability* is applied when the interactions are undirected or directed, respectively: if two users are connected (or one is reachable from the other) in a per-cycle UIG, they are considered as having interacted in that cycle. Finally, the ratio between the total number of these recognized cycles and the window size gives their interaction strength. In this way, UISE captures the frequency of both direct and indirect interactions.

Figs. 4.2 and 4.3 show the per-cycle UIGs derived from Figs. 4.1(a) and 4.1(b), respectively. In Fig. 4.2, an algorithm for finding connected components is applied. Here, users i and j have interacted directly in cycles 1 and 3, and indirectly in cycle 2. Therefore, their user interaction strength is estimated to be 0.75. Similarly, in Fig. 4.3 an algorithm for finding the reachability is applied and user j is reachable from user i in cycles 1, 2, and 3. Therefore, the user interaction strength from user i to j is estimated to be 0.75.

The connected components in an undirected graph and the reachability in a directed graph can be computed in linear time (in terms of the numbers of the vertices and edges of the graph) using either breadth-first search or depth-first search [11]. Thus, UISE achieves linear time complexity for estimating user interaction strength.

4.4.3 Maintaining security

Through UISE a user can rank other users based on its interaction strengths with them. When some privilege is give to the high-ranked ones, malicious users are incentivized to spread false information to be better off than others. One prevalent way to do so is through a *Sybil attack* [4, 20]. Under a Sybil attack, the attacker



(a) Global UIG (b) Local UIG of user i

(c) Local UIG of user j (d) Local UIG of user k

Figure 4.5: An example of the global and local user interaction graphs (UIGs)

generates multiple *sybils* with fake identities, and together they distribute false information about strong interactions among them. These forged interaction histories cause false edges to be added to the per-cycle UIGs of other users, where the attacker manages to connect to its sybils and to further connect to more users (victim users) through its sybils. In this way, the attacker potentially achieves higher estimated interaction strengths at victim users than without the sybil attack. Nevertheless, the false edges not only connect the attacker and victim users, but also victim users themselves (indirectly). Similarly, they also potentially increase the estimated user interaction strength between victim users. So, spreading false information does not make the malicious users be better off than others. In this way, UISE actually provides disincentives for malicious user behaviors.

Fig. 4.4 shows an example of a Sybil attack. Here, the attacker uses its real identity, j, to interact with user i in cycles 1 and 3, and with user k in cycle 2; it uses its fake identity, m, to interact with user k in cycle 3. Further, j and m claim that they have interacted in all 4 cycles. Because of the false edge added between j and m (represented by the dashed line), the attacker is connected to the victim user (user k) through its Sybil m in cycle 3 (Fig. 4.4(b)). As a consequence, user k is considered to have interacted with the attacker in cycles 2 and 3, while originally only in cycle 2. On the other hand, user k is now connected to user i (another victim user) through m in cycle 3. It is considered to have interacted with i in cycles 2, 3 and 4, while originally only in cycles 2 and 4.

4.4.4 Incorporating into distributed systems

We now show how UISE can be incorporated into a distributed system without central servers, which comes down to the question of how a user obtains interaction histories of other user pairs and builds its local UIG, and to what extent that a local UIG resembles the global one.

We assume that in a distributed system, users can obtain information through dissemination. In UISE, after two users have directly interacted with each other for the first time in a particular cycle, they generate an *interaction record*, and disseminate this record into the system through the dissemination protocol provided by the system. Based on the interaction records obtained through dissemination, each user builds its own *local* UIG, which represents its local view of the system. Further dividing a local UIG based on cycles gives the *local per-cycle* UIGs. A local UIG is a subset of the *global* UIG, in terms of the vertices, the edges, and the interaction bitmaps. The global UIG can only be obtained when the underlying dissemination protocol achieves a 100% coverage. In this paper, the 100% coverage case is used as the baseline for performance evaluation in Sections 4.7, 4.8, and 4.9.

Fig. 4.5 shows an example of the global and local UIGs. Notice that user i did not receive the interaction record between j and k for cycle 3. Therefore, in its local UIG (Fig. 4.5(b)), the interaction bitmap between j and k is 0100, instead of the ground truth 0110. A similar loss of information also happens to j (Fig. 4.5(c)) and k (Fig. 4.5(d)).

Until now, we have introduced the basic design of UISE. In the following sections, we demonstrate and evaluate two examples of its application. In Section 4.5, we apply it to detect user interaction patterns in Facebook, a centralized online social network. In Sections 4.6-4.9, we apply it to derive a decentralized scheme for estimating the online times of users and we evaluate its performance through simulations, emulations, and Internet deployment.

4.5 Interaction pattern detection

In this section, we apply UISE to detect user interaction patterns in online networks. There have been several works addressing this issue [2, 19, 21, 22], but they only consider direct and binary user interactions. Instead, UISE captures the

frequency of both direct and indirect interactions, and therefore provides a more thorough indication of the nature of user interactions. We take Facebook as an example and we apply UISE to detect its user interaction patterns. As a result, we have derived patterns that highly resemble the ones often observed in the offline human society.

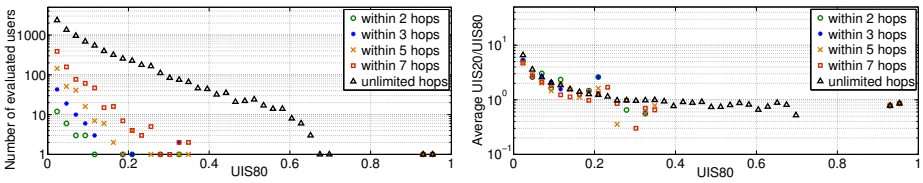
4.5.1 Applying to interaction pattern detection

We take *wall posts* in Facebook as the example of user interactions. In Facebook, each user can post messages on the walls of his friends. Wall posts are visible to others who visit the user profiles. A previous measurement study [19] contains a snapshot taken on Jan 22, 2009, of the entire wall post histories of 60,290 users in the New Orleans network. We select their data of the last year—from Jan 23, 2008 to Jan 22, 2009—as the user behavior imported into our experiment. In total, 44,397 users and 876,993 posts are included.

We set the cycle size to one week and we divide the data into two parts: we take the first 80% as the training data (weeks 1-43), and the latter 20% as the testing data (weeks 44-54). When a message is posted on a wall in some week, the two friends involved are considered as having directly interacted, and the corresponding bit in their interaction bitmap is set from 0 to 1. As Facebook is centralized, we can obtain the interaction bitmaps of all user pairs. For every user pair, we evaluate their user interaction strength (UIS) based on the two parts of the data, and we refer to the results as UIS80 and UIS20, respectively. We use UIS80 to demonstrate the user interaction pattern and we use the ratio between UIS20 and UIS80 (represented as UIS20/UIS80) to demonstrate the evolution of the user interaction. The results are shown in the following section.

4.5.2 Results

We first show the UISs of a highly active and a medium active user with all users with whom they have interacted, either directly or indirectly. The highly active user has been active in 53 out of 54 weeks and has exchanged 2,083 messages with 25 of his friends; the medium active user has been active in 26 out of 54 weeks and has exchanged 84 messages with 8 of his friends. For each of these users (called the evaluating user) we calculate his UIS80 and UIS20 with all other users



(a) The number of evaluated users in each UIS80 group

(b) Average UIS20/UIS80 in each UIS80 group

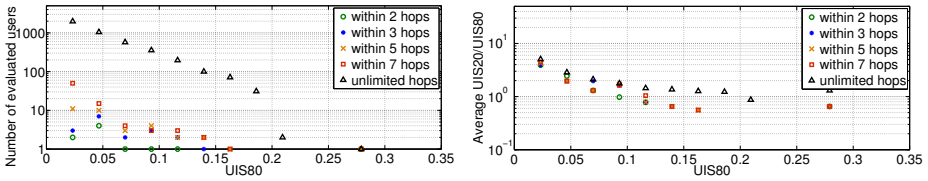
Figure 4.6: Interaction pattern of a highly active Facebook user (the vertical axes are in log-scale)

(called the evaluated users), including the friends he has interacted with directly. We group the evaluated users based on the value of their UIS80.

For the highly active user, we show in Fig. 4.6 the number of users and the average UIS20/UIS80 for each UIS80 group (represented by the black triangles, we do not consider a limit to the distance between interacting users here). While this evaluating user has only interacted directly with 25 friends, UISE links him to more than one thousand users through indirect interactions, among which, obviously, it interacts intensively with only a small group: as shown in Fig. 4.6(a), the number of users in the UIS80 groups decreases dramatically with increasing values of UIS80. A similar phenomenon is often observed in human society where people tend to interact frequently with relatively small groups and occasionally with the persons outside those groups. The small group could be friends, with whom people interact directly, or friends of friends, with whom people build bonds through, for example, sharing gossips with friends.

Another interesting observation is that, as shown in Fig. 4.6(b), the average value of UIS20/UIS80 decreases with increasing values of UIS80 until UIS80 equals 0.25, and stays stable (at a little bit less than 1) afterwards. This indicates that (1) the interactions between the evaluating user and the users with whom it has high interaction strengths in the first 80% of the year tend to stay stable, with slightly decreased interaction strengths in the latter 20% of the year; and (2) the interactions between the evaluating user and the users with whom it has low interaction strengths in the first 80% of the year tend to become more intense. A similar phenomenon is often observed in human society where people interact frequently and stably with some close friends and in the meantime make new friends and gradually lose touch with some old friends as well.

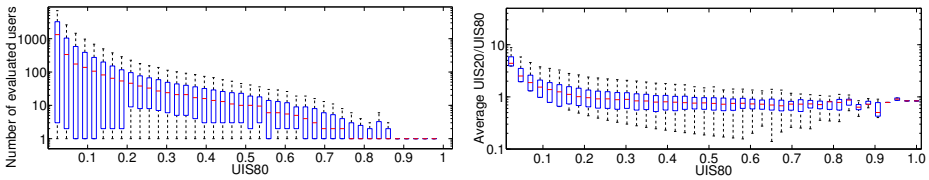
To verify whether the above observations are due to indirect interactions link-



(a) The number of evaluated users in each UIS80 group

(b) Average UIS20/UIS80 in each UIS80 group

Figure 4.7: Interaction pattern of a medium active Facebook user (the vertical axes are in log-scale)



(a) Statistics of the number of evaluated users in each UIS80 group

(b) Statistics of the average UIS20/UIS80 in each UIS80 group

Figure 4.8: Interaction pattern of all Facebook users (the vertical axes are in log-scale)

ing too many strangers to the evaluating user, we have also performed tests where we consider a limited distance in terms of the number of hops between interacting users. For example, for the result of "within 2 hops" as shown in Fig. 4.6, we have only considered the evaluating user's friends and friends of friends. We show the results for different numbers of hops, and we observe a similar tendency as for our original design without a hop limit.

For the medium active user we show in Fig. 4.7 the number of users and the average UIS20/UIS80 for each UIS80 group. In general, this user achieves a similar interaction pattern as the highly active user, except that his interaction strengths with other users are always less than 0.4—indicating that he is indeed a user with medium activity.

We have tested all users as in the above two examples and in Fig. 4.8 we show the minimum, the maximum, the median, and the 25th and 75th percentiles of the number of evaluated users and of the average value of UIS20/UIS80 of all users (there is no limit to the distance between interacting users here). We find similar user interaction patterns in these results as in the two examples we gave earlier.

4.6 Distributed online time estimation

In this section, we introduce another application of UISE, which is distributed online time estimation. Online time directly reflects user activity and is therefore important for online networks. A potential utilization in Facebook (especially in a distributed version) is evaluating user's stickiness by estimating the time users spend being online. And in BitTorrent, as users upload when they download, online time implies a user's contribution level and therefore can be used to design incentive policies [7]. The advantage of online time is that it is a metric with a ground truth—by comparing the real and the estimated online times, we can assess the accuracy of UISE. In this section, we introduce how to apply UISE to derive a decentralized scheme for online time estimation, and how to implement this application into Tribler. In Sections 4.7-4.9, we evaluate its performance by means of simulation, emulation, and real world deployment.

4.6.1 Applying to distributed online time estimation

Similarly as we previously took wall posts in Facebook as the example of user interactions, in this application we take *rendezvous* as the example of user interactions. By applying UISE, a user can now estimate the online time of another user through calculating their interaction strength, i.e., the frequency of their direct and indirect interactions. Here, when two users meet (rendezvous) in a particular cycle, they generate an interaction record and disseminate it into the system. Based on the records received through dissemination, a user i builds its local per-cycle UIGs, which are undirected since rendezvous is undirected, and if in any of these a user j is in the same connected component as i, i recognizes j to be online in that cycle. The number of these recognized cycles gives j's online time as estimated by i, and the user interaction strength between i and j computed as the number of these cycles divided by the length of the interaction history, gives j's fraction of online time as estimated by i.

The requirement for being in the same connected component is for maintaining the security against malicious users, as in the example shown in Fig. 4.4, which generates a side effect of UISE—the accuracy of an evaluating user's estimations is limited by its own online time: as we will see, the more active a user is, the more accurate his estimations of the online times of others will be.

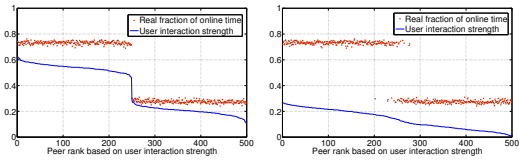
Figure 4.9: The Tribler system

4.6.2 Implementing into Tribler

We have implemented the distributed online time estimation application with UISE into Tribler [13], which is a fully distributed open-source online network for media and social applications like file sharing, live streaming, video-on-demand, content searching, voting, and interest-based channels. Users in Tribler interact in various ways including rendezvous, upload, download, and additionally, commenting, replying, voting, and reporting spam in the channels they join. Tribler serves well as a general framework for implementing, testing, and analyzing user interaction related studies. It has been instrumented with monitoring capabilities to measure both system and component specific performance for design improvements. In this section we give a brief overview of the design features of Tribler relevant for this paper.

Fig. 4.9 shows the general architecture of the Tribler system. To capture user behavior across sessions, Tribler assigns each user a permanent identifier. It uses the BitTorrent protocol for P2P file sharing, and it uses the Libswift protocol [12], which is an IETF (Internet Engineering Task Force) standard protocol for streaming proposed by the Tribler group, for P2P streaming. We have implemented UISE into Tribler as a separate component for estimating user interaction strength. The estimations can be fed back to applications for policy design, such as in file sharing to reciprocate active users with priority for future downloads; they can also be visualized in the user interface to psychologically motivate users to contribute.

Tribler uses the following protocol to discover new users and to disseminate interaction records. Every 5 seconds, a user i contacts one of its neighbors, for example, j. First, i and j generate an interaction record for this rendezvous. Secondly, j introduces one of its own neighbors to i for later contacts. Finally, i sends a Bloom filter expressing the interaction records it currently possesses and fetches the ones it is missing from j. In this way, each Tribler user can build its local UIG and estimate the online time of other users.



(a) An evaluating peer from class A (b) An evaluating peer from class B

Figure 4.10: Comparison between the real and estimated fractions of online time

4.7 Simulation

In order to evaluate the performance of UISE, we take distributed online time estimation as the example and we address three questions: how well does UISE perform for different information dissemination protocols; how well does it perform for different user behaviors; and how well does it perform in the real world.

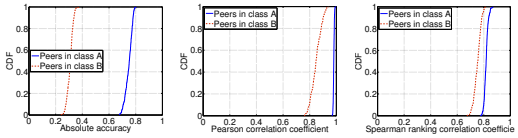
To answer the first question, in this section we run simulations with generic information dissemination, which allow us to explore the accuracy and scalability of UISE under different dissemination protocols by tuning the coverage. To answer the second question, in Section 4.8 we run emulations of UISE under various synthetic and real-world user behaviors. To answer the last question, in Section 4.9 we report measurement results derived from the Internet-deployed Tribler system.

4.7.1 Basic simulation model

Synthetic user behavior: We consider synthetic user behavior in our simulation. At any time, a peer[1] can be either online or offline. When an online session ends, it starts an offline session immediately, and vice versa. The online and offline session lengths follow exponential distributions, as observed in many distributed systems [5]. We do not consider population turnover in synthetic user behavior. Instead, later in Sections 4.8 and 4.9 we use user behavior generated from measurements where population turnover is naturally included.

Let (S_{on}, S_{off}) represent the average online and offline session lengths. We consider two classes of peers: (i) active peers, class A, with (8, 2) cycles, and (ii) less active peers, class B, with (2, 8) cycles.

[1]From here, we use *user* and *peer* alternatively to refer to the functioning agent in our experiments.



(a) Abs. accuracy (b) PCC (c) SRCC

Figure 4.11: CDFs of the absolute accuracy, PCC, and SRCC in a simulation with 500 peers.

Synthetic peer discovery and record dissemination: In our simulation, we abstract peer discovery into a constant probability, P_1, and we apply it in every cycle to specify the probability that any two online peers meet and generate an interaction record. Similarly, we abstract record dissemination into another constant probability, P_2, and we apply it in every cycle to specify the probability of an interaction record being received by a third peer. Real-world protocols are more complicated and they normally result in dynamic probabilities. Nevertheless, by assuming a constant probability and setting different values to it, we can analyze the performance of UISE for different peer discovery and dissemination protocols.

Simulation setup: We run each simulation for 336 cycles, i.e., 168 hours (7 days) when one cycle represents 30 minutes in the real implementation[2]. Unless otherwise stated, we consider 250 peers in class A and 250 in class B, and we set P_1 to 20% and P_2 to 50%.

Based on the synthetic user behavior and record dissemination, each peer gradually collects interaction records and builds its local UIG. At the end of the simulation, it estimates its user interaction strength with every other peer, which, as specified in Section 4.6.1, is equal to its estimation of the fraction of online time of another peer. By comparing this estimation with the real fraction of online time, we evaluate the accuracy and scalability of UISE. The results are presented in the following sections.

4.7.2 Accuracy

We first show in Fig. 4.10 the comparison between the real and estimated fractions of online times, where the latter is represented by the user interaction strengths

[2]We will give the reason later in Section 4.8.3

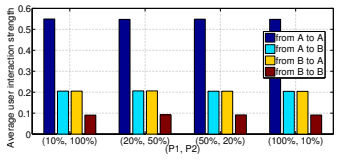


Figure 4.12: Average user interaction strengths for different values of P_1 and P_2.

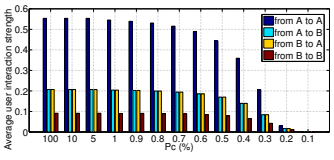


Figure 4.13: Average user interaction strengths for different values of P_C

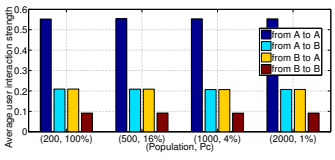


Figure 4.14: Average user interaction strengths for different populations and values of P_C

(UISs) between the evaluating and evaluated peers. In Figs. 4.10(a) and 4.10(b), the evaluated peers are ranked according to their UISs with an evaluating peer in class A and in class B, respectively. We see that the estimation is improved when the evaluating peer is more active: the peer in class A achieves more accurate estimations than the peer in class B. Peers in class B only stay online for $S^B_{on}/(S^B_{on} + S^B_{off}) = 20\%$ of the time and therefore they meet few peers to build their local views. As stated in Section 4.6.1, this is a compromise for maintaining the security.

Absolute accuracy: We define the absolute accuracy achieved by a peer as the ratio between its estimation and the real online time of another peer, averaged over all evaluated peers. In Fig. 4.11(a) we show the CDF of the absolute accuracies achieved by peers in class A and in class B, respectively. In general, peers in class A achieve better absolute accuracies than peers in class B.

Relative accuracy: We use the Pearson Correlation Coefficient (PCC) [14] and the Spearman Ranking Correlation Coefficient (SRCC) [15] to assess the relative accuracy achieved by each peer. In brief, PCC and SRCC measure the linear and the monotonic dependence between two variables, respectively. For each evaluating peer, first, we rank its evaluated peers based on its online-time estimations for them; then, we generate two variables, a list of the real online times and a list of the online-time estimations, in the order of this ranking; finally, we calculate the PCC and the SRCC of these two variables. In this way, we can assess the correlation between the local rank of peers at the evaluating peer and the global rank of peers based on their real online times. For the two examples shown in Figs. 4.10(a) and 4.10(b), the PCCs are 0.9848 and 0.9026, and the SRCCs are 0.7968 and 0.7812, respectively. Figs. 4.11(b) and 4.11(c) show the CDFs of the PCC and the SRCC achieved by peers in class A and in class B. We see that peers achieve decent relative accuracies, i.e., their local ranks of peers resemble the global one.

Absolute accuracy versus relative accuracy: One important application of online time estimation is to differentiate users with different levels of activity. Then, only the rank of users is needed and a design with high relative accuracy, like UISE, will be suitable. Further, as we will show in the next section, UISE gives accurate estimations even under low coverages of peer discovery and information dissemination.

4.7.3 Accuracy under partial information

To test the accuracy of UISE under partial information, we first vary P_1 and P_2 in such a way that $P_C = P_1 \times P_2$ is constant (equal to 10%). P_C represents the probability of establishing an edge between two peers in a local per-cycle UIG of a third peer. Intuitively, it decides the third peer's estimations for others. In Fig. 4.12 we show the user interaction strengths (estimated fractions of online time) averaged over the classes of evaluating ("from" in the figure) and evaluated ("to" in the figure) peers. We find that, consistent with our intuition, the estimations stay stable for different values of P_1 and of P_2 while P_C is constant. This allows us to analyze the influence of P_C without exploring extensive combinations of P_1 and P_2. We keep P_1 constant (equal to 1) and vary P_2 in such a way that P_C is decreased from 100% to 0.1%. In Fig. 4.13 we show the user interaction strengths averaged over classes. We find that when P_C is at least equal to 5%, i.e., when peers hold at least 5% of the total information, the estimations stay stable. Further reducing P_C results in noticeable decreases of estimations, nevertheless, UISE still achieves decent relative accuracies: on average, peers in class A are correctly estimated as more active than peers in class B.

4.7.4 Scalability

In this section we test the scalability of UISE under different populations. Let $N(t)$ represent the number of online peers at cycle t. At the end of the simulation, for cycle t, each peer will receive $N(t)(N(t)-1)P_C$ records and will generate $N(t)(N(t)-1)P_C$ edges in its local per-cycle UIG. To test the scalability, when the population increases, we decrease P_C in such a way that $N(t)(N(t)-1)P_C$ is constant.

In UISE, an evaluating peer recognizes another peer to be online at cycle t if

they are in the same connected component in its local per-cycle UIG for cycle t. In graph theory, for a random graph with n vertices to be connected, the expected number of edges needed is less than $n \ln n$ [11]. Therefore, the basic condition for a peer to correctly recognize all the peers online at cycle t is:

$$N(t)(N(t)-1)P_C \geq N(t)\ln(N(t)) \Rightarrow P_C \geq \frac{\ln(N(t))}{N(t)-1}. \tag{4.1}$$

As $\ln(N(t))$ increases very slowly with $N(t)$, the required value for P_C decreases strongly with the population. Thus, UISE achieves good scalability in a self-organized manner.

The simulation result confirms the above analysis. Fig. 4.14 shows the average user interaction strengths (estimated fractions of online time). We see that while we increase the population and decrease P_C accordingly, UISE achieves stable estimations.

4.8 Emulation

In this section we evaluate the performance of UISE for different user behaviors. As we have tested different dissemination protocols in Section 4.7, we now release the assumption of a generic information dissemination and we test UISE under Tribler's dissemination protocol. As we intend to test different user behaviors that we cannot control in real world experiments, we use emulation in this section. Based on synthetic user behaviors, we test the performance of UISE for different online patterns. Based on user behaviors generated from measurement traces, we test its performance across different online networks.

4.8.1 Emulation setup

The emulation is performed on an anonymous cluster that contains 23 nodes, each of which has two 2.4 Ghz quad-core processors and 24 GBytes of memory. The nodes are connected by a 10 Gb/s QDR Infiniband interconnect. Unless otherwise stated, we deploy 500 peers evenly on 20 nodes. Peers run Tribler's dissemination protocol: they meet, generate, and disseminate interaction records, and store the records received from dissemination in their local SQLite databases. At the end of each emulation, they estimate their user interaction strengths with

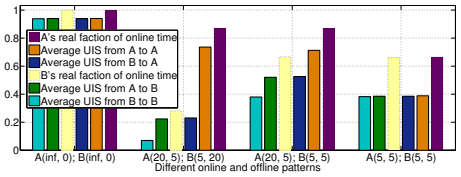
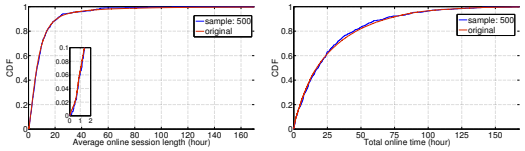


Figure 4.15: Comparison between real and estimated fractions of online time for different churn patterns (S_{on}, S_{off})



(a) CDF of session length (b) CDF of total online time

Figure 4.16: Comparison between the original FileList trace and our sample.

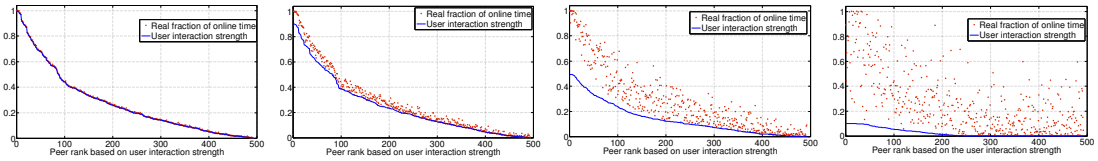
others, which give their estimations of the fractions of online time of others.

We set the cycle size to 2 minutes and we run each emulation for 10 hours, resulting in interaction bitmaps of $10 \times 60/2 = 300$ bits. The small cycle size and short emulation time are compromises for the time consumption of the cluster. This parameter setting represents a running time of 7 days when the cycle size is set to 30 minutes in the real world implementation.

4.8.2 Synthetic user behavior

In this section we evaluate the performance of UISE under different mixtures of active peers, less active peers, peers with heavy churn, and peers that always stay online. Here, we use the synthetic user behaviors as introduced in Section 4.7.1 and we test four scenarios in our emulation: (i) peers always online with A(∞, 0) and B(∞, 0); (ii) active peers A(20, 5) versus less active peers B(5, 20); (iii) active peers A(20, 5) versus heavy churn peers B(5, 5); and (iv) heavy churn peers with A(5, 5) and B(5, 5). All the session lengths are in minutes. The results are shown in Fig. 4.15.

Under scenario 1, peers have the highest chance to meet and hence generate a large number of records. Fig. 4.15 shows that UISE achieves accurate estimations under this scenario: on average, 93.33% of the online time is successfully identified. Under scenario 2, the active peers achieve higher accuracies than the less active peers. Interestingly, comparing scenarios 2 and 3, when the less active



(a) Evaluating peer always online

(b) Evaluating peer with an availability of 90%

(c) Evaluating peer with an availability of 50%

(d) Evaluating peer with an availability of 10%

Figure 4.17: Comparison between real and estimated fractions of online time for FileList trace

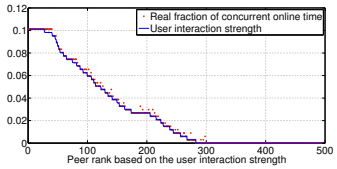


Figure 4.18: Comparison between fraction of concurrent online time and user interaction strength.

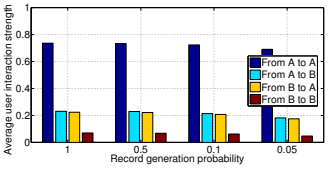


Figure 4.19: User interaction strength under random-generation for different values of P_G.

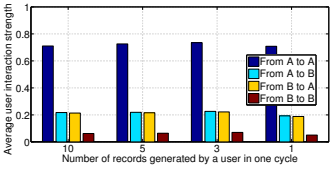


Figure 4.20: User interaction strength under targeted-generation for different values of N_G.

peers are changed to peers with heavy churn, their estimations for active users ("from B to A" in the figure) become more accurate. Under scenario 4, though peers are with heavy churn, they can still identify $0.4/0.6 = 67\%$ of the online time. These results indicate that the absolute accuracy depends on the *availability*, i.e., the fraction of online time, of the evaluating peer, rather than its churn pattern.

4.8.3 Trace-based user behavior

To test the performance of UISE across different online networks, we run emulations based on measurement traces generated from the private BitTorrent community FileList [1]. These traces contain uptime and downtime of every user that was online at least once during the measurement period. In total, we captured 63,548 users in 7 days, from which we randomly select 500 users for our emulations. Fig. 4.16 shows the CDFs of the average online session length and the total online time for the original trace and our sample. We see that our sample represents the original trace very well. Further, as 94% of the online sessions are longer than one hour, we set the cycle size to 30 minutes in the real implementation, so as to capture most of the online sessions.

Fig. 4.17 shows four examples of comparisons between the real and estimated fractions of online time, where the latter one is represented by the user interaction strengths between the evaluated and evaluating peers. In Figs. 4.17(a), 4.17(b), 4.17(c), and 4.17(d), the evaluated peers are ranked based on their online times estimated by a peer with an availability of 100%, 90%, 50%, and 10%, respectively. The first three peers achieve estimations very close to the real fractions of online time, with an SRCC equal to 0.9998, 0.9945, and 0.9388, respectively. We can see a clear decrease of the real fractions of online time when the evaluated peers are ranked based on the estimations from these three evaluating users, indicating that their local ranks of peers closely resemble the global one.

Evaluating peer 4 (Fig. 4.17(d)), however, only achieves an SRCC equal to 0.6853. The reason is that, as stated in Section 4.6.1, in UISE an evaluating peer only trusts the interaction records that can link back to itself (reflected by being in the same connected component in a local UIG). Therefore, its estimation for another user in fact reflects their *concurrent* online time. As evaluating peer 4 is only online for 10% of the time, it achieves a low accuracy. Nevertheless, its estimation for another user can be used to assess their availability to each other—an important issue for distributed online networks where users collaborate and only the ones online simultaneously can help each other. In Fig. 4.18 we show evaluating peer 4's estimations and its fraction of concurrent online time with other peers, where we observe very accurate estimations with an SRCC equal to 0.9973.

4.8.4 Preparing for the real world: strategies for reducing the workload

As we found in Section 4.7.3, UISE achieves good estimations even with limited information. This desirable feature allows us to introduce two practical strategies, *random-generation* and *targeted-generation*, to reduce the workload imposed on the system, in terms of the number of interaction records to be disseminated. In random-generation, when two online peers meet for the first time in some cycle, with a probability P_G they generate an interaction record (in the original design they will for sure generate a record). In targeted-generation, for each cycle, a user only generates records with the N_G users that it has observed to be online the longest during the past M cycles, resulting a constant number of records being

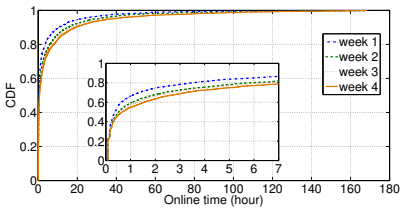


Figure 4.21: CDF of online time of Tribler users

generated per user per cycle.

We run emulations to test the performance of these two strategies, where we use the same parameter settings as in Section 4.8.2. Figs. 4.19 and 4.20 show the user interaction strengths (estimated fractions of online time) averaged over classes for different values of P_G and N_G (we set $M = 1$ in our emulations). We see that UISE performs stably when P_G is decreased from 1 to 0.05 and when N decreases from 10 to 1. This implies that we can decrease the workload dramatically without deteriorating the accuracy of the estimation.

4.9 Real-word deployment

We have implemented UISE into Tribler. In addition to its qualitative impact on design, Tribler's user community serves as the basis for our real-world performance evaluation. In this section, we report measurements of the user behavior, the overlay structure, the performance of interaction record generation, and the accuracy of online time estimation.

4.9.1 Deployment techniques

In the real world deployment, the cycle size is set to 30 minutes and the interaction history is kept in a sliding window fashion with a window size of 7 days, resulting in interaction bitmaps of $7 \times 24 \times 60/30 = 336$ bits. We adopt the *targeted-generation* version of UISE as introduced in Section 4.8.4, with $N_G = 5$ and $M = 1$. In addition, we specify that two users generate their first interaction record only until they have seen each other online for at least two cycles. This effectively prevents "hit-and-run" users generating records that are of limited use.

Tribler is fully distributed, containing no central servers and hence no records of user behaviors from the global view. To obtain the ground truth for our exper-

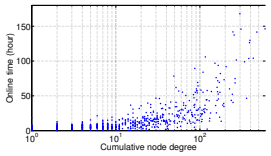


Figure 4.22: Online time versus cumulative node degree

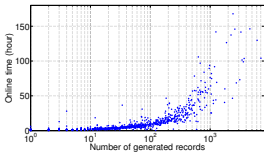


Figure 4.23: Online time versus the number of generated records

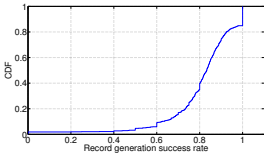


Figure 4.24: CDF of record generation success rate

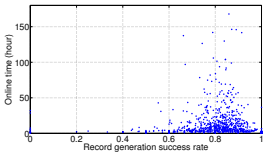
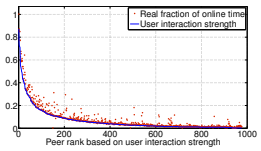
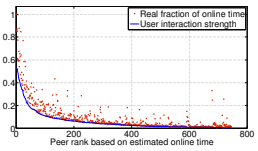


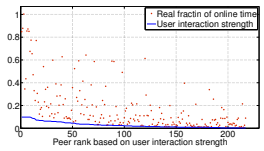
Figure 4.25: Online time versus record generation success rate



(a) Evaluating user with an availability of 90%



(b) Evaluating user with an availability of 50%



(c) Evaluating user with an availability of 10%

Figure 4.26: Comparison between real and estimated fractions of online time

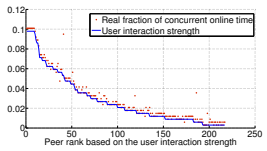


Figure 4.27: Fraction of concurrent online time versus user interaction strength

iment, we deploy log servers and every 5 minutes, each user reports its online activity to one of them, including its identifier, its timestamp, the number of interaction records it generated successfully, and the updated information about interaction records of other user pairs it received since last report. In total, for the first week of Tribler's new release, we obtain 2,874 active users with unique identifiers, among which there are 1,713 users that have generated at least one interaction record. For weeks 2, 3, and 4, we obtain 2,673, 2,905, and 2,884 active users with unique identifiers, respectively. We use the data from weeks 1-4 to analyze the evolution of Tribler user's online time; and we use the data of week 1 to evaluate the overlay structure, the performance of record generation, and the accuracy of online time estimation. Results are shown in the following sections.

4.9.2 Evaluation

Online time: The dashed blue line in Fig. 4.21 shows the CDF of online times of Tribler users obtained from log servers during the first week. Around 15% users are online for more than 7 hours, resulting in an average of more than one hour per day. Nevertheless, 60% users are online for less than one hour in total. Comparing Figs. 4.21 and 4.16(b), clearly users in FileList are more active than

users in Tribler. FileList specifies that users with high contribution levels will be rewarded with the preference for future downloads, and therefore users are incentivized to stay online longer. To do so, FileList constantly monitors user activities through central servers—UISE achieves exactly the same goal, and moreover, is performed in a distributed manner. As a matter of fact, in the current release of Tribler, users are educated with the fact that their activities will be evaluated through a distributed algorithm (i.e., UISE). Though at this moment the estimated online times are not utilized explicitly as in FileList, we can already observe a gradual increase of user's online time from week 1 to week 4 (Fig. 4.21). This promising observation indicates that, being aware of their activities being evaluated, users in Tribler are becoming more committed—a behavioral change that has been observed by many sociologist and psychologists under similar circumstances in human society [6].

Overlay structure: When users are online, they gradually meet more users and generate more interaction records. This effect is further amplified by targeted-generation where users that stay online longer are preferred by other users to generate interaction records with. In Fig. 4.22 we show the scatter plot of each user's online time and its cumulative node degree, which is defined as the number of unique users that it has generated interaction records with. We see a clear positive monotony trend between them, achieving an SRCC of 0.7883. In Fig. 4.23 we show the scatter plot of each user's online time and the number of interaction records it generated successfully, where we observe a positive monotony trend with an SRCC of 0.9312.

Interaction record generation: In Fig. 4.24 we show the CDF of record generation success rate, which is defined as the ratio between the number of generated records and the number of record generation attempts, for each user. We see that in general, 80% of the users achieve success rates larger than 0.7. The failures may come from several circumstances including the churn of users, the recipients being saturated by requests, and the NATs between users that prevent them to connect. One may argue that with the preference to generate records with highly active users, those users can be fully occupied and therefore resulting in low record generation success rates. Nevertheless, we show in Fig. 4.25 the scatter plot of each user's online time and its record generation success rate. The corresponding SRCC is only -0.1591 and therefore shows no correlation between

these two metrics. Particularly, a highly active user that stays online for 151 hours generates 2,410 records, achieving a success rate of 85.92%.

The accuracy: Fig. 4.26 shows the comparisons between the real and estimated fractions of online time of each Tribler user, where the latter is represented by the user interaction strengths between the evaluating and the evaluated users. In Figs. 4.26(a), 4.26(b), and 4.26(c), the evaluated users are ranked based on their user interaction strengths with a user with an availability of 90%, 50%, and 10%, respectively. Similar to the results of Filelist trace relay in Section 4.8.3, the accuracy of the estimation is limited by the availability of the evaluating user: the more active it is, the more accurate its estimations will be. In total, the three evaluating users have successfully identified 976, 745, and 217 users to be online for at least one cycle; and the SRCCs between their estimations and real online times are equal to 0.9325, 0.8635, and 0.6446, respectively. Though evaluating user 3 (Fig. 4.26(c)) achieves a low accuracy, it can accurately estimate its concurrent online time with other peers. In Fig. 4.27 we show its estimations (represented by user interaction strengths) and its real fraction of concurrent online time with others, where we observe very accurate estimations with an SRCC equal to 0.9785. Thus, it can successfully identify the users with whom it is online simultaneously, i.e., the users with whom it has been and potentially will be collaborating.

4.10 Related work

To date, a few works have focused on understanding user interactions in online social networks. Moon *et al.* [2] investigate the guestbook logs of Cyworld and they show that interactions between friends are highly reciprocated. Viswanath *et al.* [19] study the evolution of user interactions in Facebook and they find that user interactions change rapidly over time. These observations provide the foundation for designing UISE that only considers recent user interactions. Wilson *et al.* [21] introduce an interaction graph. They show that interaction links exhibit different properties than social links (friendships) and are more representative for inferring meaningful user relationships. Nevertheless, their interaction graph is unweighted and does not take the interaction frequency into account as we do.

Another direction of related research is identifying social ties. Kahanda *et al.* [8] propose an approach for identifying the weak and the strong ties. They focus

on supervised learning models that require human annotation of link strength such as top friend nomination. Xiang *et al.* [22] develop an unsupervised model that represents a range of tie strengths based on user interactions and profile similarity. However, they consider only direct and binary interactions, simply indicating whether a user has interacted with another user or not. Instead, we propose UISE that captures the frequency of both direct and indirect interactions. Moreover, while all the above related works are centralized, UISE is applicable in distributed systems.

There are also studies on leveraging user interactions in distributed online networks for policy design. BitTorrent [3] clients constantly monitor their direct interactions (uploads) with others and reciprocate the ones from whom they download the fastest. However, in BitTorrent systems the problem of low rendezvous exits and direct interactions are insufficient for inferring user relationships [7]. Meulpolder *et al.* introduce BarterCast, a distributed reputation system that ranks users based on their upload and download activity in P2P file sharing. BarterCast captures both direct and indirect user interactions, however, it adopts a MaxFlow-based algorithm with a heavy complexity. Instead, UISE adopts an connect-component-based algorithm and achieves a linear time complexity in terms of the number of user pairs. We have also applied UISE to derive a decentralized scheme for online time estimation. To the best of our knowledge, this is the first work that sheds lights on this topic.

4.11 Conclusion

User interaction is the most important underpinning of online networks, in which hundreds of millions of users communicate, interact, and share their online lives. In this paper we propose UISE, a scalable scheme for estimating user interaction strength in both centralized and distributed online networks. We have applied UISE to detect user interaction patterns in Facebook based on wall posts, and we have derived patterns that resemble the ones often observed in the offline human society. We have further applied UISE to design and deploy a decentralized scheme for online time estimation based on rendezvous. In the latter application we shown that UISE is scalable and stable for different dissemination protocols and for different user behaviors. We have incorporated this application

into Tribler, and we have shown through measurements that UISE effectively differentiates users with different levels of activity, and thus, accomplishes the most important goal of estimating user interaction strength.

As future work, we plan to apply UISE to more types of interactions than only wall posts to detect interaction patterns and rendezvous to estimate online times. For instance, we intend to explore chats over photos to friendship recommendation and upload and download to incentive policy design.

Bibliography

[1] P2P Trace Archieve. http://p2pta.ewi.tudelft.nl/pmwiki/.

[2] H. Chun, H. Kwak, Y.H. Eom, Y.Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in colum vs interaction: a case study of cyworld. In *Proceeding of IMC'08*, 2008.

[3] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, USA, 2003.

[4] J. Douceur. The sybil attack. In *Proceeding of IPTPS'02*, 2002.

[5] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, 2005.

[6] B.A. Jacob. Accountability, incentives and behavior: The impact of high-stakes testing in the chicago public schools. *Journal of public Economics*, 89-5:761–796, 2005.

[7] A.L. Jia, R. Rahman, T. Vinko, J.A. Pouwelse, and D.H.J. Epema. Fast download but eternal seeding: the reward and punishment of sharing ratio enforcement. In *Proceeding of IEEE P2P'11*.

[8] I. Kahanda and J. Neville. Using transactional information to predict link strenth in onlince social networks. In *Proceeding of ICWSM'09*.

[9] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceeding of WWW'03*.

[10] S. Milgram. The small world problem. *Psych. Today*, 2:60–67, 1967.

[11] R. Motwani and P. Raghavan. Randomized algorithms. *Cambridge University Press*, 1995.

[12] R. Petrocco, J. Pouwelse, and D.H.J. Epema. Performance analysis of the libswift p2p streaming protocol. In *Proceeding of P2P'12*, 2012.

[13] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M.R. van Steen, and H.J. Sips. Tribler: A social-based peer-to-peer system. *Concurrency and Computation: Practice and Experience*, 2008.

[14] J. L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1), 1988.

[15] C. Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15, 1904.

[16] N. Tran, J. Li, L. Subramanian, and S. S. Chow. Optimal sybil-resilient node admission control.g. In *Proceeding of INFOCOM'11*, 2011.

[17] N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-resilient online content voting. In *Proceeding of NSDI'09*, 2009.

[18] A. Vespignani. The familiar stranger: An aspect of urban anonymity. *The Individual in a Social World: Essays and Experiments*, 1977.

[19] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *ACM SIGCOMM WOSN*, 2009.

[20] B. Viswanath and A. Post. An analysis of social network-based sybil defenses. In *Proceeding of SIGCOMM'10*, 2010.

[21] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Proceeding of Eurosys'09*.

[22] R. Xiang, J. Neville, and M. Rogati. Modeling relationship strength in onlince social networks. In *Proceeding of WWW'10*, 2010.

[23] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *IEEE Symposium on Security and Privacy '08.*, 2008.

[24] H. Yu, M. Kaminsky, P.B. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attacks via social networks. In *Proceeding of SIGCOMM'06*, 2006.

Chapter 5

Community building, continuation and roadmap

This section discussed the current state of the code and the best way forward. Parts of the codebase are over 10 years old and numerous developers contributed to it. Both volunteers and scientists from EU projects contributed over the years. An informal analysis of the problems and possible solutions are presented.

Key question is *how to turn the QMedia effort into a sustainable volunteers-only initiative*.

5.1 Code refactoring

This is the current status of our codebase:

We have Tribler, using an obsolete torrent engine from bittornado, which depends on an old patched VLC version, Swift and Dispersy. We als have SwarmPlay, an unmaintained fork from Tribler which has some features that would be nice to merge back to the main project.

We propose the following changes:

Separating Tribler into TriblerCore and TriblerUI

This would allow:

- For other projects to directly use all the features presently available on Tribler for alternate projects (either alternative frontends Tribler for Android,

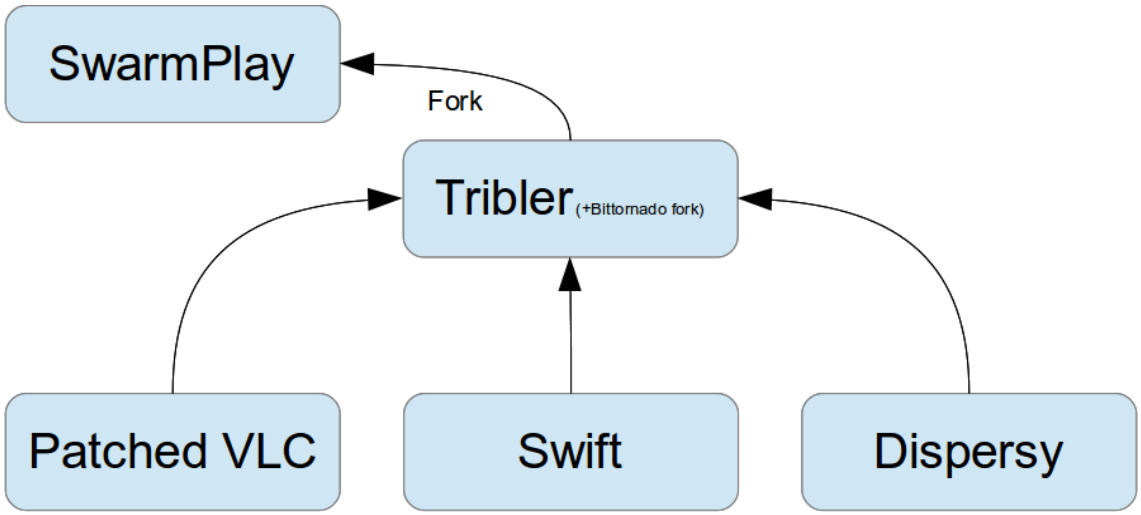


Figure 5.1: Current software structure

Tribler daemon, webservice, webUI... or totally different ones based on the same technologies such as The Global Square project)

- To simplify unit testing (we could test both the UI and the engine separately) and ease continuous integration and the development process in general.
- Keep the code cleaner and make sure no interactions between threads can exist. (IE UI blocking engine threads or the other way around)

Refactor the current code to remove Bittornado legacy code and migrate to a libtorrent based solution.

This would mean that we need to implement some extra piece picking algorithms libtorrent currently lacks (for streaming, real time broadcasting, etc...) but would allow us to:

- Have a state of the art, high performance Bittorrent engine (C++, long lived, well implemented and mantained...).
- Have the latest bells and whistles for free in the bittorrent part which the current engine lacks.
- Smaller memory/CPU footprint.

- Less threading interactions.
- Further simplify the Tribler core.
- And lastly and no for that less important: to let someone else maintain this important part for us and for free.

Get rid of VLC's fork

Try to get any patches we need accepted upstream and just use the official version.

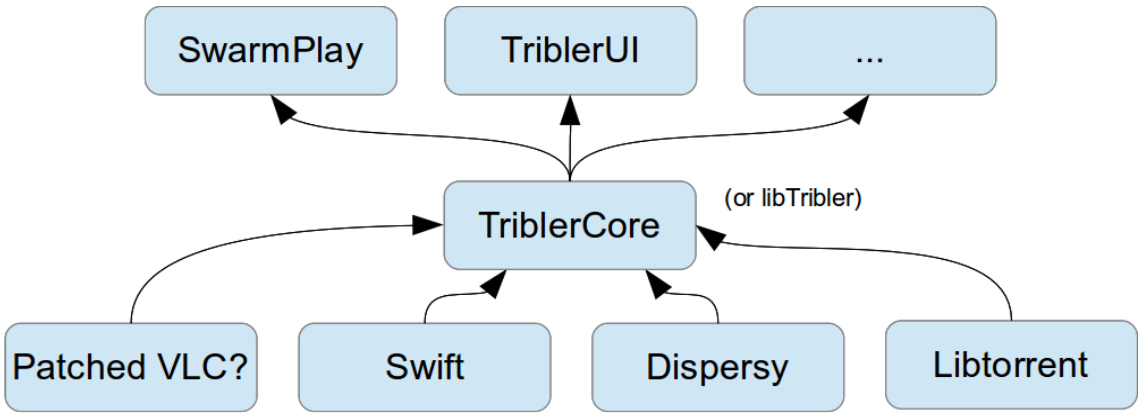


Figure 5.2: Proposed software structure

5.1.1 Other ideas to consider:

- Create an independent libCommunities which can contain a collection of pre-built Dispersy communities ready to use on other projects. This would allow to see interesting interactions between several kinds of software.
- Switch to ProtocolBuffers from google to define the community message payloads and versioning. This would allow to reduce Dispersy's code complexity, lower the bar for people wanting to create new communities. And ease alternate implementations of the protocol (as a ProtoBuf message definition can be compiled to Python, Java, C++, etc...) All this would be very interesting if we want to extend Dispersy's usage and for the IETF proposal.

5.1.2 Advantages over the current model:

- Every feature developed for parallel projects (SwarmPlay for example) is already on TriblerCore, so no need to port stuff from them back to tribler.
- Eases the development of third party projects (The Global Square for instance) as everything should be readily available from the TriblerCore API and as it's the same codebase used for Tribler it will always be in a working condition.
- All the fixes/features from both the Tribler team and the community (when/if there's any) will directly reach all the other projects with no work required (as long as the API remains stable).

Questions pending to answer:

- What are the modifications made to VLC?

5.2 Repository conversion and reorganization

I propose to switch from SVN to Git for two basic reasons:

1. If we want to create a developer community around the Tribler project we will need a distributed SCM so we don't have to keep managing contribuitors permissions in the official repository and to ease forks and merges from forked projects.
2. If we have to switch to a distributed SCM it should be git as it's (one of) the most widely used and the one that the main social networks for developers are offering hosting for:

- GitHub only offers Git and SVN via an experimental Git-SVN bridge.
- Bitbucket both Git and Mercurial.
- Sourceforge offers SVN, Git, Mercurial and Bazaar.
- Gitorious only offers Git.
- Google code offers both SVN, Git and Mercurial.

- Launchpad which only offers Bazaar support.

There are a lot of branches that aren't useful at all (old employees, personal branches for testing, etc...) I propose a controlled repository migration to GIT, converting only the relevant branches and tags with its corresponding history and keeping the SVN repo as a read-only reference and as document management system for papers and publications in general.

5.2.1 Proposed repository architecture

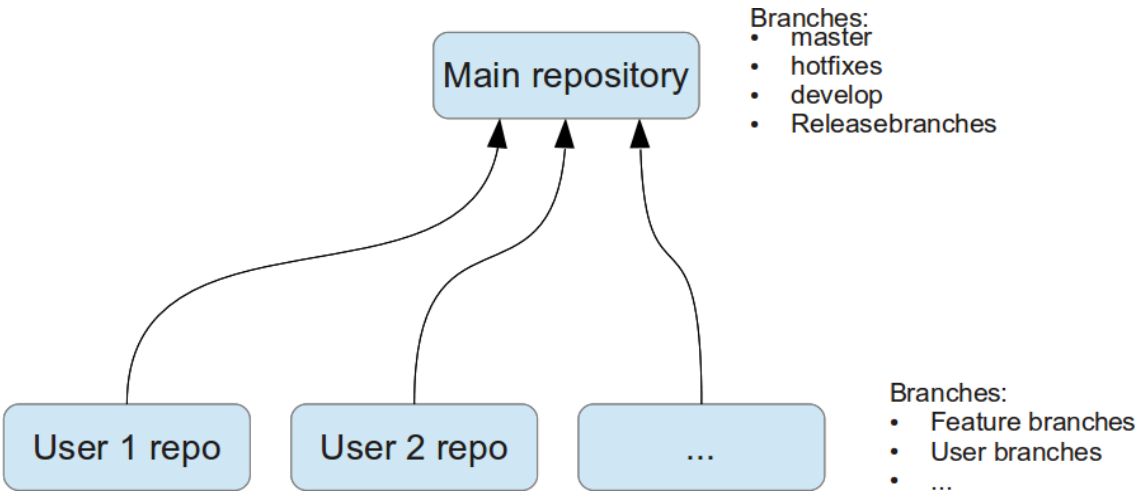


Figure 5.3: Proposed repository structure

OR, if we decide to have all branches centralized:
This is for each component (TriblerCore, TriblerUI, Dispersy, Swift...)

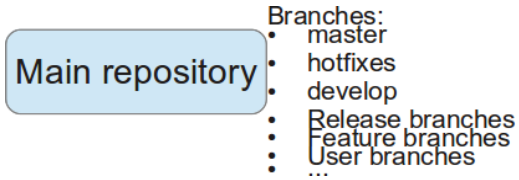


Figure 5.4: Proposed centralized repository structure

Branch names and its contents

- Master
  - Contains the latest release code

  - Each release gets merged to this branch and then a tag is added indicating the version number.
  - It only receives merges from the release branches and the hotfixes branch.

- Hotfixes
  - Only used to implement hot fixes for critical bugs in released code.
  - After the hotfix is tested this branch will get merged to the develop branch and to the master branch + tagging or release branch

- Develop
  - This is the branch where all the feature branches get forked from and merged to once they are completed.
  - This way, this branch should never have half-implemented features or WIP/untested code.
  - It gets merges from both feature branches and the hotfixes branch.

- Release branches
  - Each time a release needs to be prepared, a new release branch gets forked from the develop tree. This is where all the final testing/stabilization/packaging work gets done.
  - When the code is ready for release, the branch gets merged back to both the develop and master branch, in the later, a tag is created with the new version number.
  - Once the release branch has been merged, it gets destroyed (All the branch history will be available in both the develop and master branches).

- Feature branches
  - Every time a new feature is to be developed, a fork from the develop branch will be created.
  - Once the feature is finished and tested, the branch can be merged back to the develop branch.

  - If the feature takes long to develop and the develop branch is diverting too much, regular merges from it can be done during the branch lifetime.

See http://nvie.com/posts/a-successful-git-branching-model/ for a nice printable poster and a fully detailed description of the proposed workflow.

5.2.2 Advantages over the current model

- Strict branching/merging model which eases the mergin process.
- Only predefined branches will stay in the main repository. This means:
  - We can avoid cruft.
  - It will be much more easy for possible colaborators to see on which branch they should base their work.
- We can have a repositry at GitHub.
  - More visibility
    * Potentially more users.
    * Potentially (more) volunteer collaborators.
- We don't need to give commit permissions to everyone that contributes.
  - Nobody has to request permission to try to collaborate, anyone can clone the official repo, try to fix bugs, clean code, implement new features... and ask for a pull request. If we like it, we can merge it back, if not, our repo/history will not be altered in any way.
  - That means that a developer community can actually be created (that's almost impossible with the current status)
- Everyone can easily fork the project.
  - That means:
    * More impact.
    * Possibility to get feedback, ideas or contributions from those parallel projects.

5.2.3 Technical advantages of Git over Subversion

- Offline operation: branches can get created/deleted, changes can be commited, any point in history can be checked out, etc. Without having a working connection to the main repository.
- Stash support: you can stash out current modifications on a temporary local commit to work on other things or to switch branches if you notice you where working on the wrong one.
- Local history rewriting: This allows us to clean up a feature branch commit history before merging it back to the development branch for example.
- Easy commit and branch/tag history browsing and visualization.
- Floating modifications.

5.2.4 Disadvantages over the current model

- The current developers need to learn and get used to Git, which is more complex than SVN.
- We need to convert the current SVN history to Git.

5.2.5

Other thoughts

Thanks to Gitolite we can keep fine grained permission control in the main repo. And even have private namespaces for developers if we ever wanted to keep all branches centralized.

If we decide to migrate, it would be better to do it as soon as possible, before more developers are hired to work on the project.

Git migration is an independent step from the others, Redmine can be integrated with both Git and Subversion, and we can switch the associated repo from a Redmine project keeping everything else. (We could even have both of them associated for some time if we wanted to) And on the continuous integration side needed to switch from one system to the other should be minimal.

Possible alternatives:

SVN2Git: Privative software that keeps a Git repo in sync with a SVN, bidirectionally. It seems that it works pretty nicely and it's quite painless to set up. It's free as in beer ATM but it looks it will stop being so when it gets out of beta.

5.2.6 Release model

Public betas

I think we should have public betas and RCs. It will help our code quality while putting user's expectations on its place. The software should warn the users to set their expectations acordingly.

Be clear about what the program does (and does not)

- What's the privacy level you can currently get.
- What analisys/experiment info is being sent to our servers.
- What can our servers ask the program to do.
- ...

5.3 Continuous integration

We also need to implant a CI server to be able to automatically check the code quality and keep track of the evolution of its behaviour, performance, etc. For this, we will be using Jenkins, a mature and well supported CI server.

5.3.1 Unit testing

Python:

For python testing we will be will be using Nose[1] which extends the builtin python unittest classes, making easier to find and run tests and allows to generate several types of reports and has code coverage support. It also integrates nicely with the Jenkins continuous integration system.

[1] https://nose.readthedocs.org/

C++

For C++ the best option seems to be Googletest[2], it too allows us to also obtain code coverage reports and to integrate the run results with Jenkins too.

Java/Dalvik (Android)

We are still pending to investigate the best option for Android.

5.4 Issue tracking

To be able to track the project issues and new feature implementations we need an issue tracking system. The three best options we have are:

- Redmine[3], well mantained and modern.
- Trac[4], well mantained but a bit old.
- GitHubs's[5] integrated issue tracker, modern, well mantained and zero mantenance.

The best option would then be to directly use GiHub's service.

5.5 Code documentation

We need an automated code and API documentation system, the best one I could find is Twisted's project one: Pydoctor

5.6 API control

As Python by default doesn't support API (interface) definitions, we might benefit from using something like zope.interfaces[6] or python contract[7]

[2]http://code.google.com/p/googletest/
[3]http://www.redime.org/
[4]http://trac.edgewall.org/
[5]http://www.github.org/
[6]http://wiki.zope.org/Interfaces/
[7]http://wiki.zope.org/Interfaces/FrontPage

5.7 Other documentation

For the rest of the documentation this project is generating the best format would be TeX, as it's the standard in academic environments.

5.8 Other tools

Other tools worth taking a look at:

- Easy-git: Makes simpler to use git for begginers.
- kdiff3 and meld: Graphical merging tools, kdiff3 integrates nicely with git.
- Egit: Graphical user interface for git, allows to easily visualize the commit history in all branches at the same time and to do basic merging cherry-picking among others.

Chapter 6

Dissemination and Exploitation Plans

In the prior chapter the continuation was discussed without additional funding, in the form of a self-sustainable community.

We now discuss our attempts to find industry support.

Unfortunately, there has been only limited interest from industry on the technology we developed. Companies such as Google and Spotify have hired multiple of our graduates. Their key interest is in the talent we educate and little interest exists for utilizing QMedia technology for a commercial service.

Our activity has received public attention. Mostly due to our ability to spread content in a distributed manner, without the need of a centralized service run by a third-party, for instance Googles YouTube. This ability allows users to distribute video without restrictions and even without connection to the Internet (using phone-to-phone technology).

A representative from Delft has visited Google headquarter for initial discussions during 2012. Our December 2012 Tribler Mobile app has been shown to Google for review. A Google manager understood our technology and deemed it very cool, but no further action is expected. Discussions with Adobe have been initiated. Due to the rapid rise of HTML5, their Flash plugin strategy is in clear decline. With some efforts, we have had face-to-face discussions, including the senior engineer responsible for the P2P overlay in Adobe products. Their technology is a generation behind our solutions, as shown by the failure of Adobe Flash on smartphones. However, they do not seem interested to utilise the latest

university research output and our seamless HTML5 integration. Ties with Spotify are close and they are aware of our technology, but they show little interest in using our software. They seem to prefer incrementally improving their own P2P technology.

Therefore, creating a sustainable community using Github seems the best way forward. Over 6 EU research projects (both FP6 and FP7) contributed to the QMedia body of code. Hopefully 7th project will emerge.

Bibliography

[1] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, 00(c):9, 2000.

[2] Angela Beesley, Elisabeth Bauer, and Kizu Naoko. How and Why Wikipedia Works : An Interview. *Computers and Society*, pages 3–8, 2006.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[4] Guido Boella and Leendert Van Der Torre. Permission and Authorization in Policies for Virtual Communities of Agents. *Informatica*.

[5] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and whitewashing in peer-to-peer systems. *IEEE Journal on Selected Areas in Communications*, 24(5):1010–1019, May 2006.

[6] R Stuart Geiger. The Work of Sustaining Order in Wikipedia : The Banning of a Vandal. *Administrator*, pages 117–126, 2010.

[7] Jim Giles. Internet encyclopaedias go head to head. *Nature*, 438(7070):900–1, December 2005.

[8] Felix Halim, Wu Yongzheng, and Roland Yap. Wiki credibility enhancement. *Proceedings of the 5th International Symposium on Wikis and Open Collaboration - WikiSym '09*, page 1, 2009.

[9] Johannes Hummel and Ulrike Lechner. Social Profiles of Virtual Communities. *Dimension Contemporary German Arts And Letters*, 00(c):1–10, 2002.

[10] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. *Proceedings of the twelfth international conference on World Wide Web - WWW '03*, page 640, 2003.

[11] B Kanefsky, NG Barlow, and VC Gulick. Can Distributed Volunteers Accomplish Massive Data Analysis Tasks? In *Lunar and Planetary Institute Science Conference Abstracts*, volume 32, page 1272, 2001.

[12] Cliff Lampe, Paul Resnick, West Hall, and Ann Arbor. Slash (dot) and Burn : Distributed Moderation in a Large Online Conversation Space. *Computer*, pages 1–8, 2004.

[13] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[14] M Meulpolder, J Pouwelse, D Epema, and H Sips. Bartercast: Fully distributed sharing-ratio enforcement in bittorrent. *Delft University of Technology-Parallel and Distributed Systems Report Series*, 2008.

[15] Laura Pearlman and Carl Kesselman. A Community Authorization Service for Group Collaboration The University of Southern California Department of Computer Science The University of Chicago Division , Argonne National Laboratory The University of Chicago The University of Southern California. *Auditing*.

[16] Michael Piatek, Tomas Isdal, Arvind Krishnamurthy, and Thomas Anderson. One hop reputations for peer to peer file sharing workloads. pages 1–14, April 2008.

[17] Reid Priedhorsky, Jilin Chen, Shyong (Tony) K. Lam, Katherine Panciera, Loren Terveen, and John Riedl. Creating, destroying, and restoring value in wikipedia. *Proceedings of the 2007 international ACM conference on Conference on supporting group work - GROUP '07*, page 259, 2007.

[18] Fernanda Viegas, Martin Wattenberg, Jesse Kriss, and Frank Ham. Talk Before You Type: Coordination in Wikipedia. *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pages 78–78, January 2007.