



**QLectives – Socially Intelligent Systems for Quality  
Project no. 231200**

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

**Deliverable D4.1.3  
QLective Platform v3 - Short report**

Submission date: 2012-02-17

Start date of project: 2009-03-01

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		
<b>PU</b>	Public	<b>X</b>
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	



## Document information

### 1.1 Author(s)

Author	Organisation	E-mail
Boudewijn Schoon	TU Delft	p.b.schoon@tudelft.nl
Tamás Vinkó	TU Delft	T.Vinko@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	14 November, 2011	Starting version, template
V0.4	13 December, 2010	First draft for internal review
V0.9	15 December, 2010	Complete first draft for internal review
V1.0	17 February, 2012	Approved version to be submitted to EC

### 1.4 Document data

Keywords	peer-to-peer, distributed mass media, distributed permission system
Editor address data	p.b.schoon@tudelft.nl, T.Vinko@tudelft.nl
Delivery date	17 February, 2012

### 1.5 Distribution list

Date	Issue	E-mail
	Consortium members	QLECTIVES@list.surrey.ac.uk
	Project officer	Jose.FERNANDEZ-VILLACANAS@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

### **Technical University of Delft**

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

### **ETH Zurich**

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

### **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

### **University of Fribourg**

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

### **University of Warsaw**

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

### **Centre National de la Recherche Scientifique, CNRS**

Paris 75006,  
France  
Contact person: Dr. Camille ROTH  
E-mail: camille.roth@polytechnique.edu

### **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](http://tribler.org); and the scientific collaboration forum [EconoPhysics](http://EconoPhysics).



# Executive summary

This report accompanies and documents the version 3.0 of the QLectives Platform software. The aim of the QLectives Platform is to combine social networking, facilitation of quality and scalable peer-to-peer (P2P) technology into a next-generation peer-production platform. All the components of the QLectives Platform are (and will be) generic and re-usable as they can handle various content types (e.g. software, video, photo, text) and are not tied to a specific application domain.

The QLectives Platform version 1.0 was built on top of the already deployed and mature P2P [tribler.org](http://tribler.org) code-base. Version 2.0 expanded this work further, see Deliverable D4.1.2. Version 2.0 was centered around the development of the Distributed Permission System, which we name **Dispersy**. Dispersy provides the necessary primitives to implement functionalities for **decentralized groups** that share a robust, secure, and scalable set of messages and permissions.

QLectives Platform version 3.0 moved decentralized groups from lab experimentation to Internet-deployment with thousands of real-world users. Version 3.0 is not a radical departure from 2.0, but provides significant incremental evolution based on three extensive tests. We expanded QPlatform functionality, deployed the software and were able to identify several bugs and unforeseen missing functionality (e.g. the need for a message priority mechanism).

Major new functionality includes: 1) random walker of the self-organising overlay which is NAT-resilient, 2) robust algorithms for efficient gossip and exchange of information using Bloom filters, 3) improved scalability of communities using subjective sets, 4) performance analysis and emulation framework, utilising our DAS4 supercomputer, 5) message bundling for performance improvement, 6) message priority mechanism, 7) dynamic permission policies which can be used to alter community rules and regulations at any time.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>QPlatform incremental improvements</b>	<b>3</b>
2.1	Peer discovery . . . . .	3
2.1.1	Improvements . . . . .	6
2.1.2	Neighborhood size . . . . .	7
2.1.3	Measurements . . . . .	8
2.2	Bloom filter selection . . . . .	9
2.3	Message bundling for performance improvement . . . . .	12
2.4	Message priority mechanism . . . . .	12
2.5	Undo . . . . .	13
2.6	Dynamic permission policies . . . . .	14
<b>3</b>	<b>Experiments and deployment</b>	<b>15</b>
3.1	Dissemination experiment . . . . .	15
3.2	Subjective set experiment . . . . .	18
3.3	Open2Edit . . . . .	19
<b>4</b>	<b>Summary and outlook</b>	<b>21</b>



# Chapter 1

## Introduction

This report accompanies and documents the version 3.0 of the QLectives Platform software. It is implemented in the context of the QLectives project and aims to serve as a generic middleware to develop peer-to-peer (P2P) applications. The central novelty of QLectives Platform's version 3.0 is the Internet-deployed Distributed Permission System, which we name Dispersy. This module provides a platform to build up communities (see Figure 1.1 for a schematic view), for example a barter community, that gives information on how generous the peers in the system are (this example is detailed in QLectives Deliverable D.4.3.2).

We will now briefly explain Dispersy's core functionality: to enable the creation of communities. Additional details can be found in Deliverable D.4.1.2. A Dispersy community is a set of peers that share the same set of messages and permission settings. The design of Dispersy and its functionality is guided by the following design considerations:

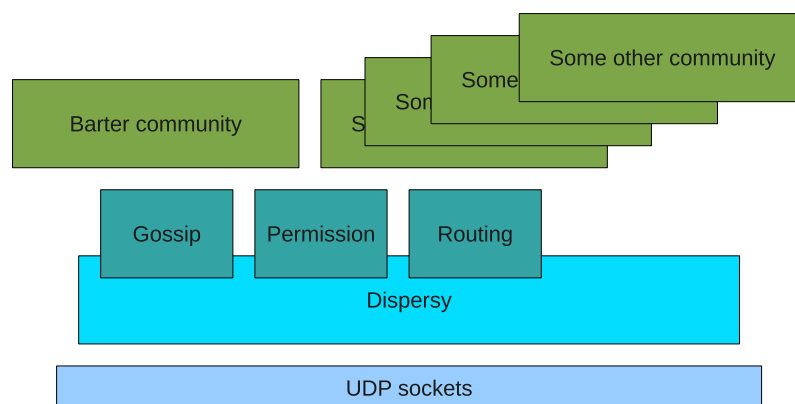


Figure 1.1: Hierarchy between Dispersy and other components.

- The first paramount factor to observe is scalability. Dispersy provides several different message policies that are designed to scale to million of simultaneous peers. While large communities will result in reduced performance, dissemination of data must always converge over time.
- Security is mild by design. The security features that Dispersy provides must be guaranteed to work. However, these features are fairly limited as security is inherently difficult to guarantee in large-scale decentralized systems.
- Security is made strong through social structures. As there is no central server that dictates policy, Dispersy creates a social structure where people higher up the chain override choices by people lower in the chain.
- Complexity, regarding the actual software development for extension of the system, should be low. Most complex actions, such as user authentication and efficient dissemination of data, should be handled by Dispersy leaving the designer of the community free to focus on end user features.

In the following chapters, we first give detailed description of all the incremental improvement of Dispersy upon the previous version, details about all the test made by means of emulations on local computer clusters. Finally, we present experiments made by Internet deployment and how they helped in further performance improvements.

# Chapter 2

## QPlatform incremental improvements

QLectives Platform version 3.0 moves decentralized groups from lab-only experimentation to a mature Internet-deployed platform with thousands of real-world users. This essential step brings us closer to the goal of providing QLectives Stream 1 inspired *mechanisms for cooperation* towards communities of potentially millions of Internet users.

This deliverable is specifically written such that the improvements are clear without the reader needing to be closely familiar with version 2.0 of QPlatform (which was reported in D4.1.2). Hence, this description is often at a high level of abstraction.

In the following sections details are given about the recent developments on the QPlatform.

### 2.1 Peer discovery

The efficiency of information exchange between pairs of peers depends greatly on the peer discovery algorithm. This algorithm is greatly influenced by the following factors:

**Peer trustworthiness.** Trivial peer discovery algorithms assume that all peers in the network are trustworthy. Others assume that a certain percentage of peers are malicious. Malicious peers can execute eclipse attacks to isolate

peers, from other well behaving peers, allowing them to dictate everything the victim receives.

**Churn resilience.** Churn is the technical term for peers joining and subsequently leaving the network. When the churn is high, i.e. peers join and leave frequently, the peer discovery algorithm needs to be fast enough to keep track of the changes, or risk propagating out-dated information.

**NAT compatible.** Many computers nowadays are behind a Network Allocation Table (NAT). This results in situations where a peer may not know the port number where other people can provide it with data, or where the NAT refuses to deliver any data without specific conditions being met.

**DDoS proof.** Distributed Denial of Service attacks are a widely known problem [7]. A peer discovery algorithm can be very susceptible to DDoS attacks when it provides a so called multiplication between the cost for the attacker versus the cost for the target. A peer discovery algorithm where these costs are equal is effectively DDoS proof.

Based on these considerations we designed an algorithm that can 1) remain secure in a network where up to half the peers are malicious, 2) will detect churn within 60 seconds, 3) can puncture through most of the known NAT configurations, and 4) is unusable for DDoS attacks.

This algorithm, called *Random Walk* (see Figure 2.1) consists of the following steps that are repeated once every 5 seconds.

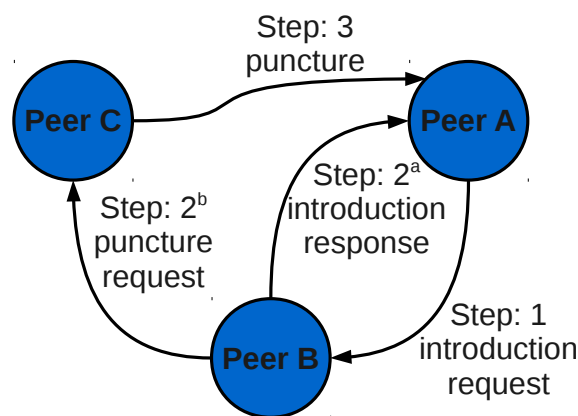


Figure 2.1: Peer discovery using a random walk.

**Step 1.** Peer  $A$  sends an introduction request to peer  $B$ . Choosing peer  $B$  is a delicate matter, it not only influences how many –potentially– new peers we might meet, but chosen incorrectly it will also facilitate malicious peers to perform an eclipse attack.

Therefore we obtain three different sets of peers (see Figure 2.2). Set  $W$  contains peers that we have previously walked too and are considered safe, set  $S$  contains peers that have contacted us, and set  $I$  contains peers that were introduced to us. Malicious peers can easily influence set  $S$  by contacting the victim frequently and set  $I$  by only introducing malicious peers.

When choosing a peer, the safest option would be to always choose from set  $W$ , however, this excludes all new peers. Therefore, we will contact a peer from set  $W$  50% of the time and otherwise we will evenly choose a peer from the remaining, potentially corrupted, sets.

**Step 2a.** Peer  $B$  sends an introduction response back to peer  $A$ , this introduces peer  $C$  to peer  $A$ , peer  $C$  is now eligible to be chosen the next time Step 1 takes place.

Peer  $C$  is introduced by providing its LAN and WAN addresses. Having both addresses will allow peer  $A$  to judge whether it is in the same LAN as peer  $C$  so that the LAN address can be used to communicate.

**Step 2b.** Peer  $B$  sends a puncture request to peer  $C$ . Peer  $C$  should respond by sending a puncture message to peer  $A$ , in the next step. If peer  $B$  knows

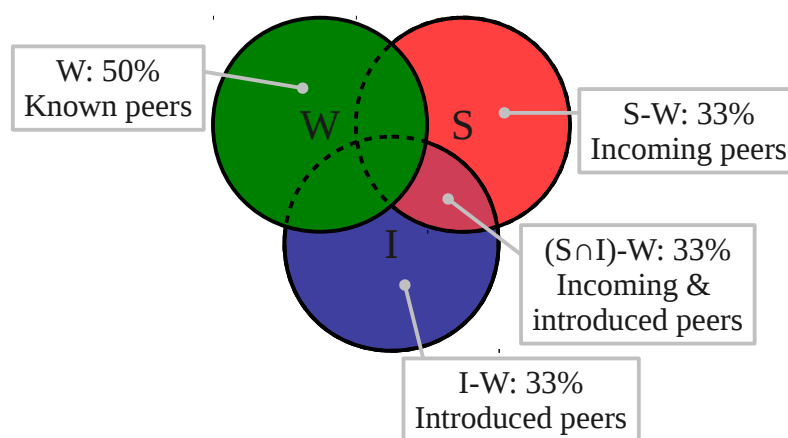


Figure 2.2: Peer selection distribution.

that peer  $C$  is not behind any NAT, the puncture request does not need to be sent.

**Step 3.** Peer  $C$  sends a puncture message to peer  $A$ . This puncture message allows peer  $A$  to determine the port number that it can use to communicate with peer  $C$  (if this is different than the one peer  $B$  knows in the case of a symmetric NAT). If peer  $C$  knows that it is not behind any NAT, the puncture message does not need to be sent.

### 2.1.1 Improvements

Concerning the problem space as it is specified at the beginning of this section, the Random Walk algorithm provides the following improvements upon the mechanism used in the QPlatform version 2 (as it was reported in D4.1.2).

The peer selection performed in Step 1 provides protection against untrustworthy peers. Malicious peers can still increase the chance that they are chosen, however, they will not be able to obtain any guarantees until a substantial part of the network has already been compromised.

As for churn resilience, since we are using UDP packets to communicate we do not have the benefit of being alerted when a traditional TCP connection is broken. Therefore, we need to actively remove peers once we have not heard from them for  $N$  seconds, where  $N$  is highly influenced by timeouts chosen in common NAT hardware and software. NAT boxes commonly [4, 5] remove an entry after it has not seen any incoming traffic for more than 60 seconds. Also, when no outgoing traffic has been detected, the NAT entry is removed after, as early as, 30 seconds. To meet with these NAT dictated timeouts, we chose to remove peers that are in set  $I$  after 25 seconds, and we remove peers that are in sets  $W$  or  $S$  after 50 seconds, measured from the last time we last received data from the associated address.

NAT puncturing is inherently solved by always involving three peers in the random walk algorithm, we always have a known third party that facilitated the introduction to a –potentially– new peer. This allows us to puncture all NAT barriers, except when both peer  $A$  and peer  $C$  are behind a symmetric NAT.

Finally, because we chose to introduce only a single peer with each introduction response, we eliminate the multiplication factor that makes DDoS attacks

possible. Therefore, in our system, the cost for both the attacker and victim is the same. Hence, an attacker could just as easily directly attack the victim.

## 2.1.2 Neighborhood size

Given that we remove peers from our neighborhood when we have not received any messages from them within, as explained in the previous section, the last 25 or 50 seconds, the size of our neighborhood follows the following function:

$$N(d, n) = 2 \left( \frac{n - \frac{5}{6} \frac{50}{d} \frac{1}{6} \frac{25}{d}}{n} + \frac{n - \frac{4}{6} \frac{50}{d} \frac{2}{6} \frac{50}{d}}{n} + \frac{3}{6} \frac{50}{d} \right) \quad (2.1)$$

$$= \frac{275 d n - 4375}{3 d^2 n}. \quad (2.2)$$

Where  $N$  is the average neighborhood size,  $d$  is the frequency of the random walk in seconds, and  $n$  is the network size. Our current implementation runs the random walk algorithm once every 5 seconds, resulting in an average neighborhood consisting of  $N(5, n) \approx 18$  peers as  $n$  approaches infinity

Figure 2.3 shows the approximate neighborhood sizes for different random walk frequencies as  $n$  approaches one million peers.

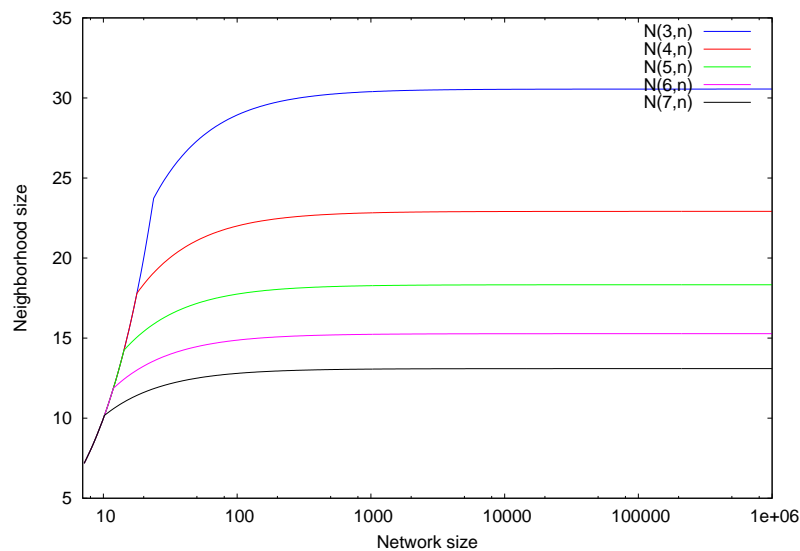


Figure 2.3: Approximate neighborhood sized give different random walk frequencies.

Peer	Found	NAT specifics		
		Brand	Model	Firmware
300	28			
302	25	D-Link	DIR-615	v4.0, Oct 22, 2008
304	25	Linksys	WRT-320	v1.0.03, Jul 24, 2009
305	27	SMC	WBR14S-N4	v0.0.0.6, Nov 26, 2009
306	28			
307	25	D-Link	DIR-300	v2.04, Mar 2, 2010
308	28	D-Link	DIR-100	v1.12, Apr 25, 2008
309	5	Sweex	LW140	v3.23, Oct 17, 2005
310	28			
312	25	Linksys	WRT54GL	v4.30.7, Jun 20, 2006
313	23	3Com	3CR858-91	v1.08, May 15, 2006
314	28	D-Link	DIR-652	v1.0
315	18	Linksys	WRT120N	v1.0.01
316	28			
317	28			
318	28			
319	25	Asus	RT-N12	v1.0.0.6
320	28	Netgear	WNR1000	v1.0.1.1
321	28	Netgear	WGR613v10	
323	20	Belkin	N300	v1.00.15, May 13, 2010
324	28	Belkin	N150	v1.00.3
325	20	Belkin	F5D5231-4	
326	25	Conceptronic	C100BRS4H	v2.19.0001
327	26	Conceptronic	C150BRS4	v2.08
331	28			

Table 2.1: Random walk in NAT conditions.

### 2.1.3 Measurements

As a preliminary study we measured the performance of the random walk algorithm by running 100 instances spread evenly across 10 peers of the DAS4 [2] supercomputer. After 15 minutes all peers communicated at least once with, average, 73 other peers. The peer with the smallest neighborhood communicated with 62 peers. On average every peer used approximately 1 MB of bandwidth during the experiment, resulting in an average of 1.1 KB/s.

Our second measurement was performed on the DAS2 [1] supercomputer where we deployed several different NAT devices (see Table 2.1). This test was designed to verify that the NAT puncturing behaved as expected. The test used a total of 25 peers and 5 bootstrap peers, the table shows how many unique peers

each peer connected with during a 20 minute experiment. On average each peer communicated at least once with 25 peers. A noticeable exception is peer 309 which was only able to communicate with the 5 bootstrap peers.

## 2.2 Bloom filter selection

What information to exchange between peers is one of the key difficulties for efficient group communication. Both push, pull and hybrid solutions have been studied in-depth in the past in contexts such as database synchronisation and multicast.

We use Bloom filters<sup>1</sup> to essentially compress the partial state of one peer into a single and efficient UDP packet. Hence, an incoming Bloom filter efficiently expresses what messages a peer already has. A receiving peer can find the information that the sender misses and provides those in a response.

However, Bloom filter compression only goes a certain way. The number of messages that can be represented within a Bloom filter is limited to the error rate [3] that QPlatform allows. A higher error rate will result in more false positives, which in turn blinds the receiving peer to the messages that the sender is missing. It is important to note that a higher error rate will never result in the transfer of duplicate messages, only missed opportunities of sending missing messages. Hence, a higher error rate results in slower convergence of the network.

Our current implementation uses an error rate of 1%. Given that the message containing this bloom filter may not exceed one MTU<sup>2</sup>, or 1500 bytes, the bloom filter is limited to 1270 bytes. This results in a bloom filter capacity of 1059 items per bloom filter. As such we need to choose which messages we will sync.

As we use a Lamport [6] global time based clock (see Deliverable D.4.1.2) this question can be reduced to selecting a time interval. We define the following two scenarios, and note that these scenarios are in direct opposition of each other: (1) a peer joins an existing community and needs to fetch all existing messages, we call this *catchup*, and (2) a peer already has the entire history and is only interested in obtaining newly created messages, we call this *update*. The catchup scenario benefits from selecting evenly across the time space. The update scenario only

---

<sup>1</sup>Bloom filters [3] are a compact representation of a set of items. It uses a combination of  $k$  hash functions to set  $k$  bits in a bit string to 1.

<sup>2</sup>[http://en.wikipedia.org/wiki/Maximum\\_transmission\\_unit](http://en.wikipedia.org/wiki/Maximum_transmission_unit)

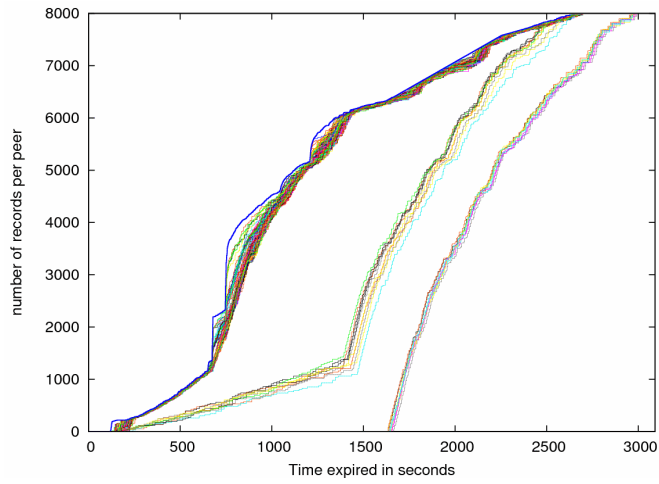
benefits from selecting at the end of the time space.

Because of the nature of distributed systems it is impossible to determine whether a peer has all existing items. Hence, it is not possible to use either strategy. Therefore we employ a hybrid strategy, that allows for both catchup and update.

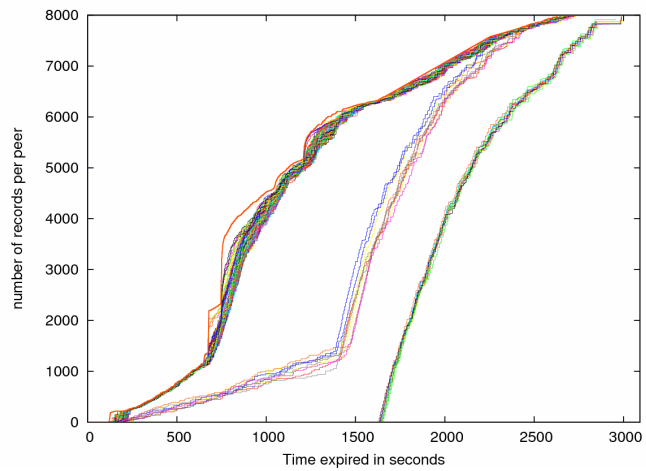
To measure the performance of different strategies we deployed 100 peers on the DAS4 [2]. One of these peers created, at variable rates, new items. These items were disseminated using the following three bloom filter selection strategies:

- A point in time is chosen using an exponential distribution. From this point we pick  $c$  more recent items, i.e. where the global time is higher than the chosen point. This strategy gives an unfair advantage to the more recent messages, however, the selection overhead is very small. The dissemination efficiency is shown in Figure 2.4(a).
- A point in time is chosen using an exponential distribution. From this point we pick  $\frac{1}{2}c$  less recent and  $\frac{1}{2}c$  more recent items, i.e.  $c$  items are used around the chosen point. This strategy does not benefit either catchup nor update scenario's, however, the selection overhead is larger than the previous strategy. The dissemination efficiency is shown in Figure 2.4(b). We found that the efficiency of this and the previous strategy are comparable.
- A point in time is chosen using an exponential distribution. From this point we pick  $c$  more recent and  $c$  less recent items. Following, this we chose either the more recent items or the less recent items based on whichever items cover the largest global time range. This strategy has the same selection overhead as the previous strategy and Figure 2.4(c) shows that the dissemination efficiency is higher than the two other strategies.

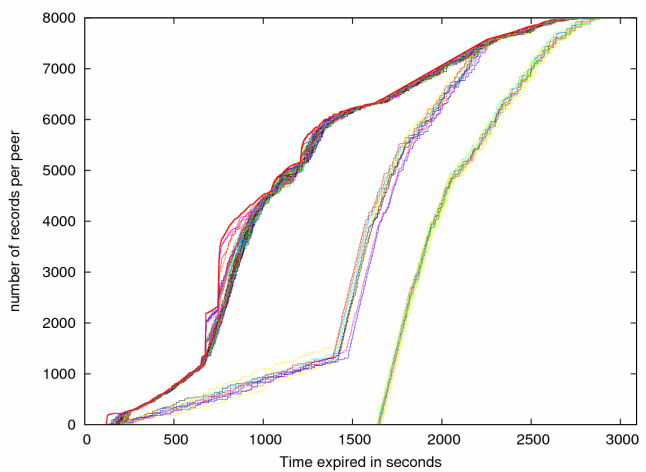
While all strategies ensure convergence, the third strategy converges faster. This is especially noticeable with the peers that join the community halfway through the experiment. Though in this experiment the convergence is improved by roughly two minutes. Bloom filter selection is ongoing work which we hope to improve further upon in version 4.0 of QPlatform.



(a)  $c$  more recent.



(b)  $\frac{1}{2}c$  more recent and  $\frac{1}{2}c$  less recent.



(c) Largest  $c$  more recent or largest  $c$  less recent.

Figure 2.4: Bloom filter range selection strategies.

## 2.3 Message bundling for performance improvement

For simplicity QPlatform version 2.0 processed each incoming message in isolation. We discovered a potential gain in performance when bundling transactions for the underlying SQL database. When similar incoming QPlatform messages are bundled and processed in a single database transaction we can reduce total processing time.

To verify our claims we generated, on one machine, 1500 different messages and process them either in a single batch or individually. Processing time is 12.20 and 19.78 seconds, respectively. Given that approximately 90% of the processing time is allocated to the verifying the cryptographic key, we see an improvement of 7.58 over 8.80 seconds, or 86%.

## 2.4 Message priority mechanism

QPlatform version 2.0 treated each message within a community equally. For some applications, performance is significantly increased when priorities can be assigned. We designed and implemented this functionality for version 3.0 using a simple *high-priority first* mechanism.

Each message is designed with two different priorities, one decides the response order, and one decides the processing order, as described below:

**Response order.** An incoming Bloom filter shows what messages another peers has, both high and low priority messages are expressed equally. When preparing a response with missing information, high priority information is provided first. Information of increasingly lower priority is provided, until the maximum response quota is reached.

This priority value is also used to exclude certain messages from being disseminated through the gossiping mechanism as only information with a higher than 32 priority is eligible for dissemination.

For example: response ordering is used to improve the dissemination efficiency of messages responsible for granting and revoking permissions.

This ordering is applied at the sender side.

**Processing order.** All incoming messages are collected, possibly in batches, and scheduled for processing at their processing priority. Messages with dependencies can thereby be reordered and processed in the most optimal order.

For example: process ordering is used to ensure that dispersy-identity messages –these messages contain the public key of a member– are processed before messages that are signed. This ensures that we do not unnecessarily request missing dispersy-identity messages.

This ordering is applied at the receiver side.

## 2.5 Undo

QPlatform version 2.0 presented the choices that were made when designing the permission system. The ability to limit the creation of harmful messages, by revoking permissions from the offending peer, was one of those choices. In version 3.0 we extend this by also allowing peers to undo their own or others' existing messages.

Obviously the inability to undo irrevocable actions did not change. Therefore, care should be taken which messages are allowed to be undone. Those that are allowed can be undone in the following two distinct ways:

- A peer can undo messages created by itself.
- A peer that has the *undo permission* can undo messages of that type that are created by someone else.

This undo permission is granted using a dispersy-authorize message. Also, the undo permission can be revoked using a dispersy-revoke message. All of this is analogue to the permission system already present in QPlatform version 2.0.

When a message is undone it will not be removed from the network. While it is be marked as undone it is no longer propagated through the Bloom filter syncing mechanism, however, when a peer receives an undo message will require the associated message before the undo is transferred. There are ongoing discussions to the merits and weaknesses of this design choice, and this will be further explored and finalized in the next iteration of QPlatform.

The Undo functionality is implemented in the recently deployed version of QMedia.

## 2.6 Dynamic permission policies

Popular online communities such as Facebook, Twitter and Wikipedia change their rules and community mechanisms on a regular basis. To facilitate a similar evolution and flexibility for distributed communities we designed and implemented dynamic permission policies. They can be used to alter community rules and regulations at any time.

For instance, in the Open2Edit community of QMedia this new functionality is used to alter the rules of chat message posting. The community owner can change settings with creating a single message.

Changing permission policies poses certain risks as certain actions may no longer be permitted after a certain point in time  $P$  when the permission change took effect. Given the nature of a distributed system, there is no guarantee that everyone will be aware of the policy change at time  $P$ . Hence, the unaware peers can create messages that will no longer be valid once the permission change becomes known. Therefore, messages that allow their permission policies to be changed must also provide an undo mechanism, as described in the previous section.

# Chapter 3

## Experiments and deployment

The aim for QPlatform is to provide production-level code that can compete with cutting edge technology, both open and closed source. With QPlatform 3.0 we have taken a major step towards this goal. While in Chapter 2 the recent improvements were all tested using emulations with computer clusters, this chapter will present several experiments that were performed in various releases and the deployment of Open2Edit, which uses QPlatform.

### 3.1 Dissemination experiment

On July 4th, 2011 we publicly released QMedia version 2.5 (named as Tribler version 5.3.8), based on the QPlatform. Within this release QPlatform played only a small role, none of the functionality visible to the user depended on QPlatform, instead it exchanged messages in the background while periodically sending progress reports back to our central server.

This allowed us to test two different types of message dissemination. The *FullSyncDistribution* is designed to disseminate each message across all peers, while the *LastSyncDistribution* is designed to disseminate only the most recent message that is created by a peer, across all peers. These policies are described in QLectives Deliverable D4.3.2.

During the experiment only a single member was allowed to create messages. One new FullSyncDistribution message, called *full-sync*, was created every hour, while a new LastSyncDistribution message, called *last-sync* was created every 3 minutes. Using the progress reports we were able to determine how many

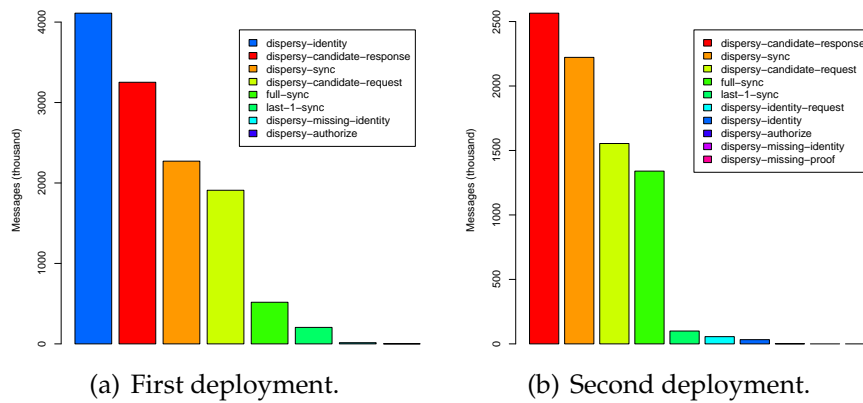


Figure 3.1: Successfully received messages during dissemination experiments.

messages were successfully received, see Figure 3.1(a). Furthermore, we could see how often –and why– incoming messages were dropped, see Figure 3.2(a).

Based on these reports we identified and improved several features and subsequently released QMedia version 2.6 (named as Tribler version 5.3.9) on July 19th, 2011. After analyzing the reports from this second experiment, see Figure 3.1(b) and 3.2(b), we determined that we solved the following issues:

**Disseminate public keys on demand.** Figure 3.1(a) shows that more than 4.1 million dispersy-identity messages were received. These messages contain a members’ public key and is required when a peer needs to verify a signature in a received message. During the first experiment this message was disseminated using full gossip, hence every member that joined the system for only a brief period would leave behind a message that had to be delivered to any and all peers in the network.

There was already a mechanism in place that would request the dispersy-identity message whenever it was required and missing, hence we decided to stop disseminating identity messages all together. For this purpose we extended QPlatform with the response ordering mechanism described in section 2.4.

The second experiment shows that approximately 33 thousand dispersy-identity messages were received. As both experiments were running under different circumstances, i.e. different churn, different duration, etc., it is not possible to compare these two values directly, however, the relative cost went down considerably. While in the first experiment dispersy-identity

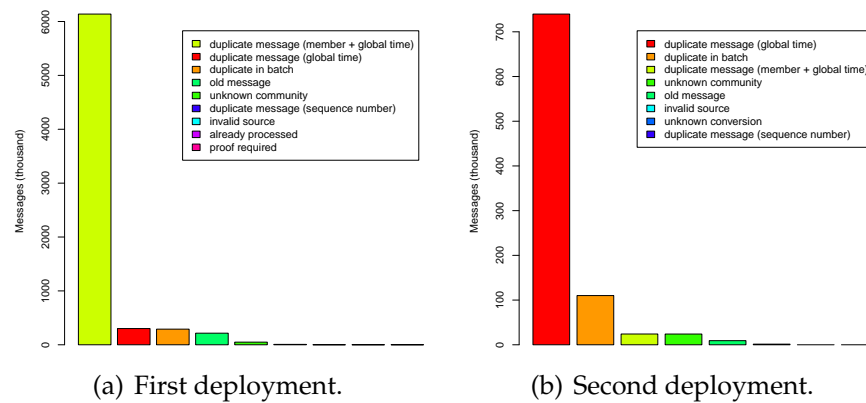


Figure 3.2: Failed message deliveries during dissemination experiments.

messages constituted for 33.5% of all received messages, this was reduced to only 0.4% after our modification.

**Fixed a bug causing duplicate messages.** Figure 3.2(a) shows that 6.1 million messages, totaling over 1 gigabyte, were dropped because they had already been received and processed. This turned out to be a bug in the bloom filter range selection and was easily fixed for the second experiment, see Figure 3.2(b), where only 24 thousand messages, or 4 megabytes, were dropped for this reason.

Note that in distributed systems it is not always possible to prevent the transmission of duplicate messages. And while 24 thousand messages may sound impressive, it pales in comparison to the 7.9 million messages that were successfully transferred during the second experiment.

**Implemented delay by proof.** Figure 3.2(a) shows that messages were sometimes dropped because of missing proof. This exception occurred when a peer received either a full-sync or a last-sync message without yet having the associated dispersy-authorize message that proved that the message creator has the required permission.

Even though this only occurred 175 times during the experiment it was important to solve this problem because it can easily occur much more frequently in communities that have more complicated permission structures and larger databases. Figure 3.2(b) shows that none of the messages are dropped because proof is required, while Figure 3.1(b) shows that these are replaced with several requests for missing proof.

### 3.2 Subjective set experiment

In an ideal world QPlatform communities would have unbounded scalability for both number of members and messages. And while current architecture should be able to scale to potentially millions of community members and numerous messages, where eventually each community member obtains a copy of each message, this may not always be desired as synchronizing all this information can take considerable time.

Hence methods need to be employed to reduce the amount of information that each peer is willing to store and share. One such mechanism is the subjective destination. In short the subjective set mechanism allows each peer to declare a set of peers that they are interested in and that will only accept messages that are created by peers that they declared in their subjective set.

This mechanism was implemented and included in QMedia version 2.6 (named as Tribler version 5.4.0), which was released on August 30th, 2011. To verify that it functions properly, we let three different peers –designated *A*, *B*, and *C*– periodically create messages. Everyone that participated in the experiment would randomly choose two of these peers and add them to their subjective set. Subsequently they should only receive the periodically created messages from the peers that they ‘subscribed’ to.

Again we gathered reports during the experiments, see Figure 3.3, from this we gathered approximately 23 thousand subjective set declarations. Each peer *A*, *B*, and *C* was chosen approximately 15 thousand times. Almost 115 thousand messages were received that were created by either *A*, *B*, or *C*, and none of the

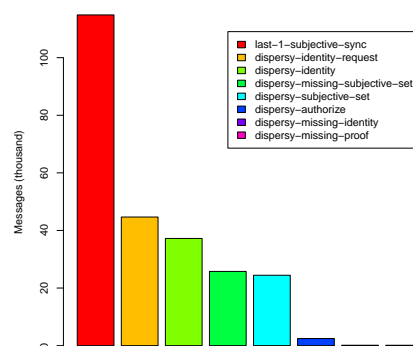


Figure 3.3: Successfully received messages during subjective set experiment. Excluding candidate-request, candidate-response, and sync messages.

sent messages violated the subjective set.

### 3.3 Open2Edit

A vital step is that our platform is now tested and used. We moved QLectives technology from the lab to *production-level code* used by real Internet communities. In version 2.0 of our platform we build upon prior Tribler work, but with 3.0 we replaced the existing foundations with technology completely developed within QLectives. Those foundations originate from 2006, when the first gossip protocol was deployed on the Internet to a large user base [8].

The Open2Edit feature of QMedia version 3.0 is the first application designed and build upon QPlatform version 3.0. Details about Open2Edit is reported in D4.3.3.

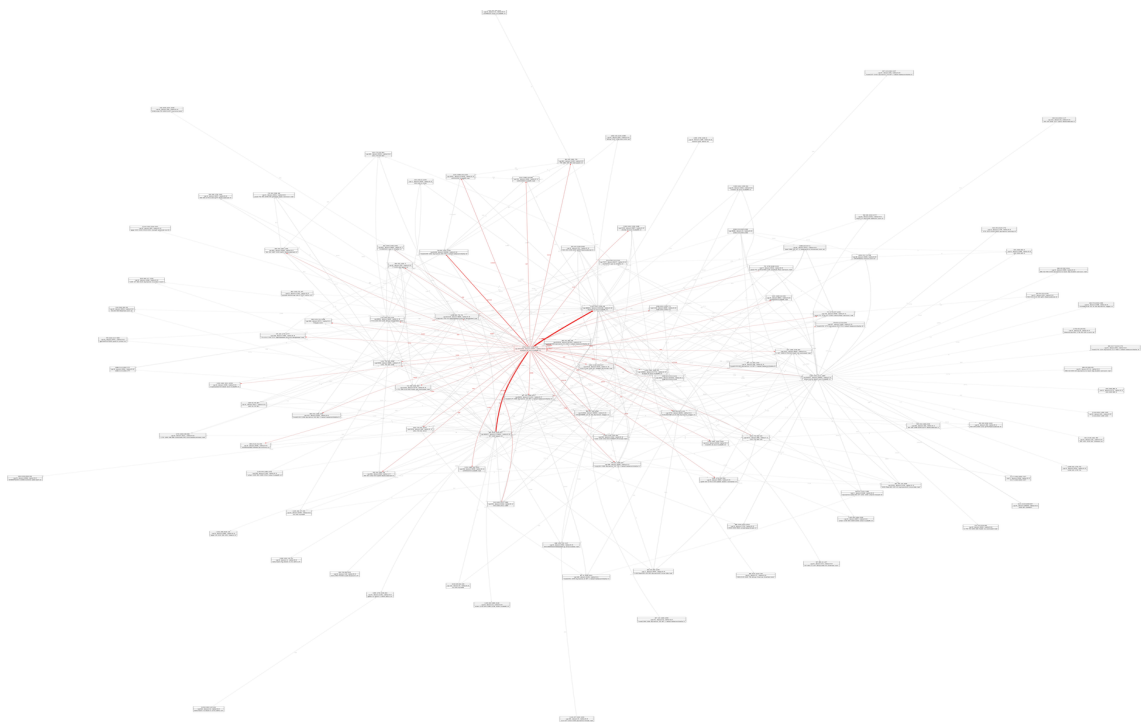


Figure 3.4: Information exchange graph from QPlatform deployment

Shown in Figure 3.4 are numerous peers running the QPlatform software and exchanging information. These results are obtained from real Internet deployment. The nodes represent peers and the weighted directed edges represent both the amount plus direction of information exchange. The density and semi-

randomness for the edges depicts a healthy exchange between pairs of peers. The center of the graph is formed by a reliable and well connected test computer. The numerous thick and thin edges connecting this peer to the rest of the graph shows the impact of single connectable and always-on peer.

During 2011 we tested QPlatform with hundreds of community members and thousands of messages to synchronize. This provided us with months of work for fixing bugs, improving performance and even expanding QPlatform with new features.

QMedia version 3.0 (named as Tribler 5.5.4) was released in early December 2011. According to the data collected at [statistics.tribler.org](http://statistics.tribler.org) it has been downloaded more than 2,200 times.

# Chapter 4

## Summary and outlook

This report documents the third release of the QLectives Platform, that can be used to base the development of different distributed communities. For that, this report describes all the incremental improvement made on the previous version of the QPlatform. Version 2.0 was built upon prior Tribler work, but with this new version we replaced the existing foundations with technology completely developed within the QLectives project. We also briefly reported Open2Edit, the first deployed application designed and build upon QPlatform version 3.

The QLectives Platform lays the foundations for experimentation with the algorithms developed in Stream 2, in particular with those reported in Deliverables 2.3.1 and 2.1.3. Moreover, the implementation described in this report is the base of the development of QMedia version 3, described in Deliverable D4.3.3.

There are several possible research lines for future works, two of them are discussed below. We have already been working on these ideas, however, they are in very early stage, results expected and will be reported in next year's deliverables.

### **Rewarding effort in online communities**

Overcoming the tragedy of the commons has been a challenge for several decades. How to effectively manage public goods and rewarding cooperative behavior has proven to be equally difficult in the digital realm.

An existential threat to self-organising systems is lack of reliable elements to build upon. When computer resources are only donated for several minutes it is becoming increasingly difficult to provide compelling and reliable services. Thus donating computer resources for an extended amount of time requires effort, but

boosts performance and reliability.

As a first step to rewarding cooperative behavior we have been experimenting with measurements of online time. Measuring online time of individual peers is not a trivial problem in a distributed setting, as others peers do the observations are themselves unreliable. The approach we have developed is based on *signed online bitmaps*.

### **Group selection for communities of cooperation**

Our most ambitious contribution is moving beyond tit-for-tat. We are hoping for a breakthrough in indirect reciprocity research. Experimentally validate that potentially millions can cooperate and manage a shared public good.

# Bibliography

- [1] The Distributed ASCI Supercomputer 2. <http://www.cs.vu.nl/das2/>
- [2] The Distributed ASCI Supercomputer 4. <http://www.cs.vu.nl/das4/>
- [3] A. Broder, M. Mitzenmacher, and A.B.I.M. Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, 2002.
- [4] L. D’Acunto, M. Meulpolder, R. Rahman, J.A. Pouwelse, and H.J. Sips. Modeling and analyzing the effects of firewalls and nats in p2p swarming systems. In Cynthia Phillips David A. Bader, Alan Sussman, editor, *Proceedings IPDPS 2010 (HotP2P 2010)*, pages 1–8. IEEE Computer Society, April 2010.
- [5] Gertjan Halkes and Johan A Pouwelse. Udp nat and firewall puncturing in the wild. In *Networking 2011, 10th International Conferences on Networking (IFIP’11)*. Springer-Verlag, Lecture Notes in Computer Science (LNCS), 2011.
- [6] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
- [7] F. Lau, S.H. Rubin, M.H. Smith, and L. Trajkovic. Distributed denial of service attacks. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 2275–2280, 2000.
- [8] J.A. Pouwelse, J. Yang, M. Meulpolder, D.H.J. Epema, and H.J. Sips. Buddy-cast: an operational peer-to-peer epidemic protocol stack. In *Proc. of the 14th Annual Conf. of the Advanced School for Computing and Imaging*, pages 200–205. ASCI, 2008.