Challenges, Designs, and Performances of Large-Scale Open-P2SP Content Distribution

Zhenhua Li, Yan Huang, Gang Liu, Fuchen Wang, Yunhao Liu, *Senior Member*, *IEEE*, Zhi-Li Zhang, *Fellow*, *IEEE*, and Yafei Dai, *Member*, *IEEE*

Abstract—Content distribution on today's Internet operates primarily in two modes: *server-based* and *peer-to-peer* (P2P). To leverage the advantages of both modes while circumventing their key limitations, a third mode: *peer-to-server/peer* (P2SP) has emerged in recent years. Although P2SP can provide efficient hybrid server-P2P content distribution, P2SP generally works in a *closed* manner by only utilizing its private owned servers to accelerate its private organized peer swarms. Consequently, P2SP still has its limitations in both content abundance and server bandwidth. To this end, the fourth mode (or says a generalized mode of P2SP) has appeared as *"open-P2SP"* that integrates various third-party servers, contents, and data transfer protocols all over the Internet into a large, open, and federated P2SP platform. In this paper, based on a large-scale commercial open-P2SP system named "QQXuanfeng" [1], we investigate the key challenging problems, practical designs and real-world performances of open-P2SP. Such "white-box" study of open-P2SP provides solid experiences and helpful heuristics to the designers of similar systems.

Index Terms—Internet, content distribution, server based, peer-to-peer, P2SP, open-P2SP

1 Introduction

1.1 Peer-to-Server/Peer

Content distribution on today's Internet operates primarily in two modes: 1) *server-based* (see Fig. 1a) and 2) *peer-to-peer* (P2P) (see Fig. 1b). Both modes have their unique characteristics and accompanying disadvantages (as described in Section 1 of the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.252). To leverage the advantages of both modes while circumventing their key limitations, a third mode: 3) *peer-to-server/peer* (P2SP) (see Fig. 1c) (e.g., [2], [3], [4], [5], [6], [7], [8], [9], [10]) has emerged in recent years. As depicted in Fig. 1c, P2SP usually utilizes a *private owned cloud* (or server cluster) as well as a number of peer swarms for content distribution. A peer swarm starts by obtaining a content seed from the cloud, and subsequently, peers within the swarm can exchange data among themselves via a *private designed P2P protocol*. Compared to the server-based mode, P2SP incurs far lower infrastructure and network bandwidth costs. Meanwhile, P2SP enhances the working efficiency of P2P by providing extra server bandwidth to those peer swarms who do not have adequate download bandwidth among them.

Although P2SP can provide efficient hybrid server-P2P content distribution with moderate server bandwidth, P2SP generally works in a *closed* manner by only utilizing its private owned servers to accelerate its private organized peer swarms. Consequently, P2SP still has its limitations in both *content abundance* and *server bandwidth*. First, for copyright and storage reasons, a P2SP system only provides a small subset of Internet contents, so its users often have to resort to other web servers, P2P/P2SP systems for contents—a quite inconvenient process. Second, due to the high dynamics of peers, it is quite possible that the server bandwidth of a P2SP system cannot satisfy its users' requirements when the user scale increases suddenly and dramatically [6].

1.2 Open Peer-to-Server/Peer

To this end, the fourth mode (or says a generalized mode of P2SP) has appeared as in Fig. 1d *"open-P2SP"* (e.g., Xunlei [11], QQXuanfeng [1], Flashget [12], and Orbit [13]), which integrates various third-party servers, contents, and data transfer protocols all over the Internet into a large, open, and federated P2SP platform as shown in Fig. 1d. The advantages of open-P2SP are mainly fourfold.

- First, open-P2SP continuously tracks and indexes ubiquitous *downloadable contents* in third-party servers all over the Internet, so as to allow end users to search and quickly find contents, and meanwhile enables peer nodes that are interested in the same content to find each other and form an *open and larger*

- *Z. Li is with the School of EECS, Peking University, the School of Software and TNLIST, Tsinghua University, 156 Chengfu Road, Beijing 100084, China. E-mail: lzh@net.pku.edu.cn.*
- *Y. Huang and F. Wang are with the Tencent Research, and the Baidu Antivirus, 690 Bibo Road, Shanghai 201203, China. E-mail: {galehuang, futurewang}@qq.com.*
- *G. Liu is with the Tencent Research, 1801 Hongmei Road, Shanghai 200233, China. E-mail: sinbadliu@qq.com.*
- *Y. Liu is with the School of Software and TNLIST, Tsinghua University, 156 Chengfu Road, Beijing 100084, China, and the Hong Kong University of Science and Technology. E-mail: liu@cse.ust.hk.*
- *Z.-L. Zhang is with the School of EECS, University of Minnesota, 200 Union Street SE, Minneapolis, MN 55416. E-mail: zhzhang@cs.umn.edu.*
- *Y. Dai is with the School of EECS, Peking University, Natural Science Building No. 1, Beijing 100871, China. E-mail: dyf@pku.edu.cn.*

Manuscript received 7 May 2012; revised 30 July 2012; accepted 14 August 2012; published online 24 August 2012.
Recommended for acceptance by K. Li.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-05-0447.
Digital Object Identifier no. 10.1109/TPDS.2012.252.

1045-9219/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

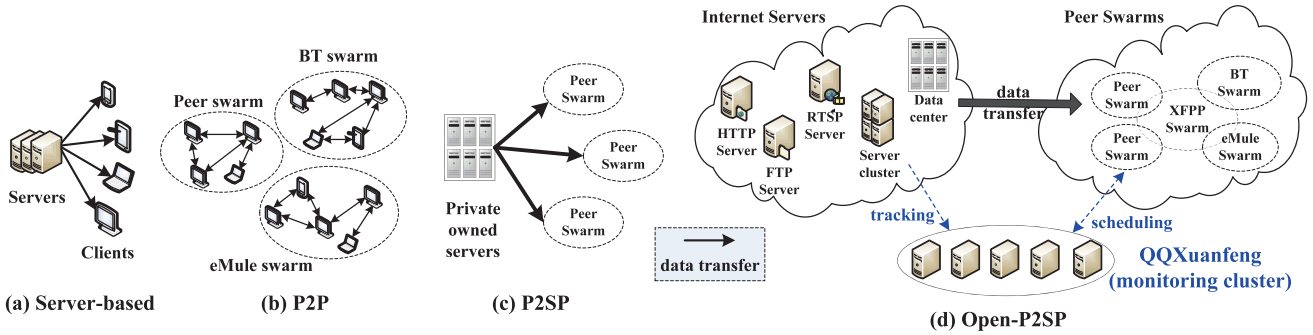


Fig. 1. General evolution of Internet content distribution modes: (a) server based, (b) P2P, (c) P2SP, and (d) Open-P2SP. "XFPP" (Xuanfeng P2P Protocol) is a special P2P protocol designed for QQXuanfeng clients. There should be numerous XFPP swarms, BT (BitTorrent) swarms, eMule swarms, and so on, while we only draw one swarm for each kind.

peer swarm. *Downloadable contents* mainly refer to videos, audios, documents, software, and so forth.

- Second, open-P2SP dynamically directs peer swarms—especially those that do not have adequate download bandwidth among them—to retrieve data from appropriate third-party servers for download acceleration. Since there are usually a huge number (typically millions) of servers involved in an open-P2SP system, it is resilient to small-scale/localized upsurge of user requirements.
- Third, by intelligently scheduling peers' data requests to appropriate third-party servers, open-P2SP also facilitates load-balancing among the involved servers while avoiding overloading a specific server.
- Finally, building an open-P2SP system does not require enormous storage servers (for storing contents) or upload bandwidth (for uploading data to peers), but merely requires a light-weight "monitoring cluster" (see Fig. 1d) for tracking servers and scheduling peer swarms. For instance, we have implemented such a monitoring cluster (named "QQXuanfeng" [1][1]) by using only 53 commodity servers. QQXuanfeng tracks over 1 million servers and serves around 5 million peers every day.

1.3 Challenges of Open-P2SP

Despite the above advantages, an open-P2SP system involves far more complicated problems than a conventional P2SP system. The major problem of designing a conventional P2SP system is handling the peer dynamics and properly allocating server bandwidth to peer swarms, while designing an open-P2SP system (e.g., QQXuanfeng) should further deal with the following key challenges:

1. *Handling server and content dynamics*. As for conventional P2SP, servers and their affiliated contents are usually stable and controllable so that its designers can focus on handling the inevitable peer dynamics. However, open-P2SP involves various third-party servers and contents with unpredictable dynamics—the involved servers never notify QQXuanfeng of their join, leave or content change.
2. *Limited utilization of server bandwidth*. A conventional P2SP system can fully utilize its server bandwidth but an open-P2SP system cannot. The *extra bandwidth* burden posed on an involved server by QQXuanfeng should be within a certain limit; otherwise, the *original service* offered by the server may be interfered. Still worse, QQXuanfeng has no knowledge of the *original bandwidth* of a server spent in its original service, which further complicates such "limited utilization."
3. *Differentiated acceleration of peer swarms*. Due to the limited utilization of server bandwidth, QQXuanfeng can hardly accelerate all its peer swarms to possess high download bandwidth. Thereby, the only choice of QQXuanfeng is to design proper differentiated acceleration strategies according to the specific requirements of different peer swarms.
4. *Bringing extra benefit to server providers*. Even if the extra bandwidth burden directed by QQXuanfeng is always restricted within a proper limit, server providers may still be reluctant to support open-P2SP if they cannot obtain extra benefit from extra bandwidth contribution.

1.4 Solutions of QQXuanfeng

Each above-mentioned problem is highly challenging as to a large-scale real-world open-P2SP system. As a matter of fact, in the past several years (since the birth of QQXuanfeng) we have (tried hard but) never figured out a *perfect* solution to any problem. Instead, our methodology is to find a *moderate and practical* solution to each problem based on comprehensive measurements. Specifically, our proposed solutions are briefly described as follows:

1. To handle server and content dynamics, we build a *content crawler* and a *content validator*. The *content crawler* continuously crawls content links (URLs) by traversing on third-party servers and receiving user reported novel links. Meanwhile, the *content validator* constantly validates the user reported invalid links by checking them on the Internet.
2. Sampling measurements indicate the *original bandwidth utilization* ("*OBU*") of an involved server usually stays below 60 percent. Thus as to each server, the *extra bandwidth utilization* ("*EBU*") directed by QQXuanfeng had better be controlled within 40 percent ($= 1 - 60\%$). Specifically, QQXuanfeng periodically collects users' reports to calculate the

1. In the remainder of this paper, we also use "QQXuanfeng" as the name of the corresponding open-P2SP system when the context is clear.

EBU of each involved server. If the *EBU* of a server S exceeds 40 percent, QQXuanfeng will notify a part of the users served by S to stop their data download from S.

3. Although every user of QQXuanfeng hopes to achieve his best user experience, through comprehensive measurements we discover that a user usually has his *basic expectation* for the download rate. Therefore, peer swarms are classified into three categories according to their real-time download rates and data supply demand conditions: 1) *hungry swarms*, 2) *potentially hungry swarms,* and 3) *high-demand swarms*. Different categories of peer swarms correspond to differentiated acceleration strategies so that each user can have a download rate at least above his basic expectation.
4. For a server provider, the two most important benefit metrics are its *page view* (*PV*) [14] and *paid-to-click* (*PTC*) [15]. At present, we are fostering benefit collaborations between QQXuanfeng and server providers from both perspectives of *PV* and *PTC*, to encourage more server providers to support QQXuanfeng.

Over the past several years (from the first client version released in 2007 to the 3.9 client version released in 2012), QQXuanfeng has gained over 120 million accumulated users and supports most mainstream data transfer protocols like HTTP, FTP, RTSP, BitTorrent, eMule, and so forth. Currently, it schedules around 5 million peers to retrieve data in petabytes from over 1 million servers per day. The average download rate of a peer is enhanced from 57 to 158 KBps, where 45 percent of the download rate is obtained from third-party servers. Meanwhile, the EBU of the involved servers is generally limited within 40 percent, so their original services are not obviously interfered. To our knowledge, this paper is the first "white-box" study of open-P2SP, which provides solid experiences and helpful heuristics to the designers of similar systems.

1.5 Roadmap

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 overviews the system design. Section 4 describes the challenging problems and corresponding solutions. Section 5 evaluates the real-world performance of QQXuanfeng. Finally, we conclude the paper in Section 6.

2 Related Work

In recent years there have been considerable researches on P2SP, such as P2SP streaming [6], P2SP storage [7], hybrid CDN-P2P [8], [9], and so forth.

2.1 P2SP Streaming

Wu et al. [6] refocused on the important role of servers in P2P video streaming. Through measurements of a popular P2P live TV system [4], they found that the total available bandwidth of the 150 dedicated streaming servers could not meet the increasing demand of download bandwidth from hundreds of TV channels (a TV channel can be seen as a peer swarm), although the total upload bandwidth of peers also increased with download demand. So they proposed an allocation algorithm of server bandwidth named Ration, which proactively predicts the minimum server bandwidth demand of each channel from historical information and thus assigns appropriate server bandwidth to each TV channel.

2.2 P2SP Storage

Based on a popular P2SP storage system [16], Sun et al. [7] discussed the tradeoff among file availability, download performance and server resource costs (including both bandwidth and storage cost) in P2SP cooperative storage. to save the critical server bandwidth cost, they concluded three conditions in which peers are allowed to obtain data from servers: 1) no online peer owns the requested file; 2) a special block of the requested file is missed in all online peers; and 3) the average download rate of the users inside a peer swarm is lower than 10 KBps. When a user uploads a file to the system, the servers make sure only one copy exists so as to save server storage cost.

2.3 Hybrid CDN-P2P

Content distribution network (CDN) like Akamai and Limelight is the most common facility that optimizes the performance of Internet content distribution by strategically deploying edge servers at multiple locations. CDN is traditionally used to accelerate server-based content distribution, but Yin et al. [8] developed a hybrid CDN-P2P video streaming system on top of ChinaCache [17] (the largest CDN in China) to incorporate the strengths on both sides: the quality control and reliability of CDN and the inherent scalability of P2P. They proposed a mechanism for dynamic resource scaling that guarantees adequate quality-of-service to end users, so as to address several shortcomings of common P2P streaming such as high buffering requirement and low streaming quality. Besides, based on the novel Akamai NetSession interface [9], Haeberlen et al. [18] designed a method for providing reliable accounting of client interactions in hybrid CDN-P2P systems.

2.4 Xunlei Open-P2SP System

Xunlei [11] is the biggest open-P2SP system in China (and perhaps in the world also). Unfortunately, till now there has been no official publication on its technical designs or system performances, but some preliminary measurements have been revealed in [19], [20], and [21] via a "black-box" method. Xunlei takes an *aggressive* data scheduling strategy which often makes nearly full use of server bandwidth to achieve high user experiences. This is a major cause of complaints of Xunlei by various server providers who have accused it of interfering with their original services. And this is one of the reasons that its initial public offerings process has been rather difficult [22]. In contrast, QQXuanfeng adopts a *more conservative* data scheduling strategy which strives for a proper tradeoff between user experience and server bandwidth burden.

3 QQXuanfeng System Overview

3.1 System Architecture and Index Structure

As depicted in Fig. 2, the QQXuanfeng system architecture consists of four major building blocks:

1. *content index DB*,
2. *content crawler*,

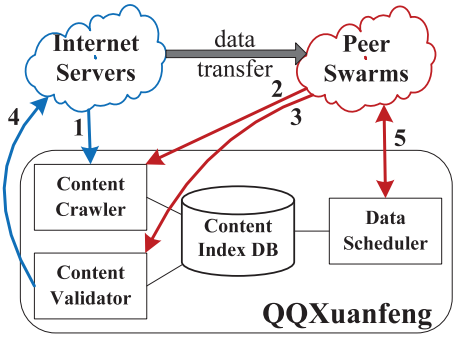


Fig. 2. System architecture of QQXuanfeng.

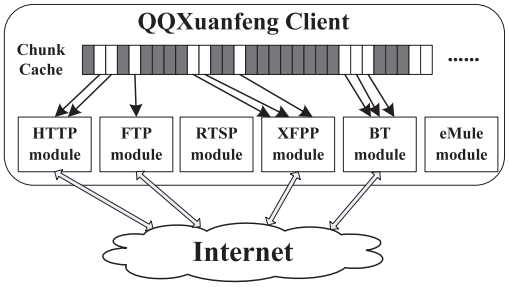


Fig. 3. Software architecture of the QQXuanfeng client. In the chunk cache, gray chunks have been obtained while white chunks have not.

3. *content validator* and
4. *data scheduler*,

utilizing 53 commodity servers in total (refer to Section 2 of the online supplementary file for detailed hardware composition). All these servers run the Enterprise Suse Linux v10.1 and gcc v4.1. The user of QQXuanfeng needs to install its client software [1] (of about 12.5 MB) which simultaneously supports the HTTP, FTP, RTSP, BitTorrent, eMule, and XFPP data transfer protocols (see Fig. 3). When installing the client, the user can choose whether to add a QQXuanfeng plugin into his web browser. If the plugin is added, the QQXuanfeng client will automatically take over the user's download task issued from the web browser.

Lying at the center of the system, the *content index DB* stores the index information of the downloadable contents detected by the *content crawler*. The content index DB employs MySQL v5.1 as its database management system. The index information includes the *(content) link*, *server IP address*, *name*, *size*, *chunk size*, *type*, *MD4 hash code*, *SHA1 hash code*, *three-chunk hash code*, and *chunk hash code list* of every indexed file. The *chunk size* should be a power of 2 bytes—typically between 32 KB and 16 MB. For an indexed file f, the 128-bit *MD4 hash code* is used for eMule data transfer because the eMule protocol employs an MD4 hash code for *content identification*. Similarly, the 160-bit *SHA1 hash code* is used for BitTorrent data transfer. Both the MD4 hash code and the SHA1 hash code are calculated (by the content publisher) from the whole content of f, but the 160-bit *three-chunk hash code* is calculated from only three chunks of f: the first chunk (C_1), the middle chunk ($C_{\lceil n/2 \rceil}$), and the last chunk (C_n) (n is the total number of chunks in f). Specifically, the *three-chunk hash code*

$$= SHA1(MD4(C_1)|MD4(C_{\lceil n/2 \rceil})|MD4(C_n)),$$

where '| means XOR operation. Finally, the *chunk hash code list* contains the MD4 hash code of every chunk.

Why is the *three-chunk hash code* necessary? When the content crawler discovers a novel eMule link, it can directly extract the MD4 hash code from the eMule link as the content identification. When it discovers a novel BitTorrent link, it first downloads the .torrent file (of several KBytes) from the BitTorrent link and then extracts the SHA1 hash code from the .torrent file as the content identification. However, when the content crawler discovers a novel HTTP/FTP/RTSP link, there is no hash code contained in the link and it is usually impractical for the content crawler to download the whole linked file (often up to hundreds of MBytes or several GBytes) to calculate the content identification. Instead, the content crawler only downloads the three special chunks (first, middle, and last chunks) of the linked file to calculate the three-chunk hash code as the content identification, and thus the resulting download traffic is affordable and the hash conflict probability is extremely low. Besides, the three-chunk hash code is also used by XFPP (the Xuanfeng P2P Protocol), a special P2P protocol designed for QQXuanfeng clients. With any kind of hash code, the content index DB can find other corresponding hash codes and a variety of links that point to the same content (i.e., the "*mapping*" operation).

The *content crawler* crawls content links by traversing on third-party servers (see Arrow 1 in Fig. 2) and receiving user reported novel links (see Arrow 2 in Fig. 2). Meanwhile, the *content validator* validates the user reported invalid links (see Arrow 3 in Fig. 2) by checking them on the Internet (see Arrow 4 in Fig. 2). With the help of content crawler and content Validator, QQXuanfeng continuously discovers novel links and discards invalid links. The detailed working principles will be presented in Section 4.1.

The basic roles of the *data scheduler* are threefold: 1) maintaining a peer list for every peer swarm (thus acting as a P2P tracker), 2) telling each user that servers and peers contain his interested content, and 3) periodically collecting users' reports to analyze their working status and calculate the EBU of each involved server (see Arrow 5 in Fig. 2). In particular, the data scheduler handles two problems: 1) limited utilization of server bandwidth and 2) differentiated acceleration of peer swarms. Their specific solutions will be elaborated in Sections 4.2 and 4.3.

3.2 A Typical User's Request Processing

Fig. 4 plots the concrete steps about how a typical user's request is processed. First, the user inputs a link (or via the QQXuanfeng web browser plugin) to the client (see Arrow 1 in Fig. 4). The user can also input several keywords and then the client will return a list of related links for a possible choice. If the input link is an HTTP/FTP/RTSP link, the client first downloads the three special chunks of the linked file to calculate the three-chunk hash code, and then sends the three-chunk hash code to the data scheduler (see Arrow 2 in Fig. 4). Likewise, if the link is a BitTorrent link, the client first downloads the .torrent file to extract the SHA1 hash code and then sends the SHA1 hash code. Besides, if the link is an eMule link, the client directly extracts the MD4 hash code from the eMule link and then sends the MD4 hash code.

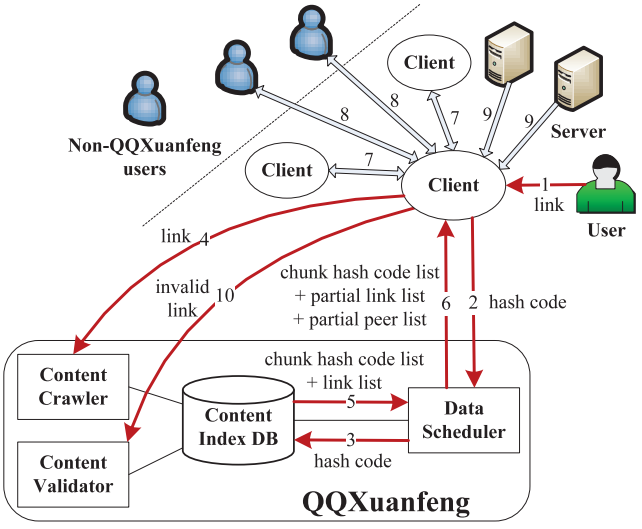


Fig. 4. A typical user's request processing.

The data scheduler inputs the hash code to the content index DB to find other corresponding hash codes and existing links that point to the same content (see Arrow 3 in Fig. 4). If the hash code is novel to the content index DB, the data scheduler will notify the client to send the input link to the content crawler for further processing (see Arrow 4 in Fig. 4). Otherwise, the content index DB *maps* the hash code to a chunk hash code list and a (server) link list (see Arrow 5 in Fig. 4).

After getting the chunk hash code list and the (server) link list, the data scheduler returns the chunk hash code list, a part of the link list (i.e., the "partial link list") and a part of the corresponding peer list (i.e., the "partial peer list") to the client (see Arrow 6 in Fig. 4). Then the client first attempts to set up data connections with the listed peers (see Arrow 7 in Fig. 4). Note that the client may also exchange data with non-QQXuanfeng users who are not monitored by QQXuanfeng (see Arrow 8 in Fig. 4). The client maintains a "chunk map" and sends the chunk map to its connected peers in every minute if the chunk map is updated. If the total P2P data transfer rate is below the user's basic expectation, the client will further set up data connections with the listed servers (see Arrow 9 in Fig. 4).

The QQXuanfeng client adopts a simple chunk scheduling strategy: at any time it only assigns one chunk for one data connection to retrieve. When the assigned chunk is obtained, the corresponding data connection will be assigned to retrieve another chunk. On the other hand, if the assigned chunk (say C_i) cannot be obtained in a "timeout period" (generally proportional to the chunk size and no less than 10 seconds) the corresponding data connection will be terminated. Furthermore, if a data connection to a server (link) is terminated for the above reason, the client will report this (possible) *dead link* (regarding to chunk C_i)[2] to the content validator (see Arrow 10 in Fig. 4). Once a chunk C_i is obtained from a link, the client validates it by using the chunk hash code list. If the client finds an *inconsistent link* (that distributes an inconsistent chunk C_i) it will report this (possible) inconsistent link to the content validator (see Arrow 10 in Fig. 4). The chunk retrieving priority depends on the user's requirement: if the user chooses the "view-as-download" mode, the chunks are retrieved in their playback order; otherwise, the chunks are retrieved in the "rarest first" manner.

2. There are two kinds of invalid links: 1) *dead links* and 2) *inconsistent links*. "Dead" means the link is inaccessible while "inconsistent" means the linked file has been changed.

4 Challenging Problems and Solutions

4.1 Handling Server and Content Dynamics

To handle server and content dynamics, we build the content crawler and content validator. The content crawler continuously crawls content links by breadth-first search (BFS) traversing on third-party servers and meanwhile receiving user reported novel links. The BFS traversing uses a *link queue* as its data structure. Various *file links* (e.g., links to videos, audios, documents, and softwares) and *nonfile links* (e.g., links to web pages) are collected by the content crawler. To avoid crawling loop links, recently visited links are cached to check whether these links have been revisited. Repeated links are discarded. Then, (possible) novel file links are directly put into the content index DB for further processing like *file link filtering* (as described in Section 5.1 of the online supplementary file) and merging into corresponding indexes. Novel nonfile links are inserted into the queue for subsequent processing.

Servers may join in or leave from the Internet for various reasons and meanwhile their affiliated contents are constantly emerging, updating, and expiring. However, the corresponding links of these contents are often left invariable, thus turning into *invalid links* (including *dead links* and *inconsistent links*) scattered all over the Internet. The negative effects of invalid links are at least threefold: 1) incurring unnecessary storage burden to the content index DB, 2) bringing unnecessary network burden to the users who try to connect the linked servers, and 3) degrading the scheduling performance of the data scheduler when it assigns some invalid links to a peer swarm for download acceleration. To this end, the content validator constantly validates the user reported invalid links by checking them on the Internet:

- On receiving a dead link l (regarding to chunk C_i) reported by a user, the content validator attempts to download C_i from l. If C_i cannot be downloaded, l is judged to be a dead link and is then removed from the content index DB. Otherwise, the content validator simply tells the reporting user that he has misreported a dead link.
- On receiving an inconsistent link l (regarding to chunk C_i) reported by a user, the content validator first downloads C_i from l and computes the latest MD4 hash code of C_i. If the latest MD4 hash code is the same as the existing MD4 hash code of C_i contained in the chunk hash code list, the content validator simply tells the reporting user that he has misreported an inconsistent link. Otherwise, l is judged to be an inconsistent link and the content validator continues to download the whole linked file, updates or adds the corresponding index information in the content index DB, and notifies the reporting user to restart his download task.

To check the practical performances of the content crawler and the content validator, we measure the numbers

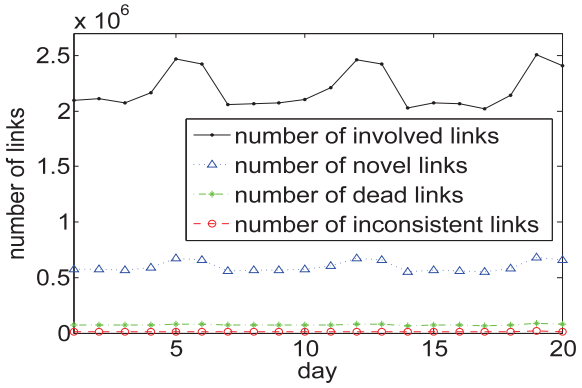


Fig. 5. Numbers of involved links, novel links, dead links, and inconsistent links in 20 days.

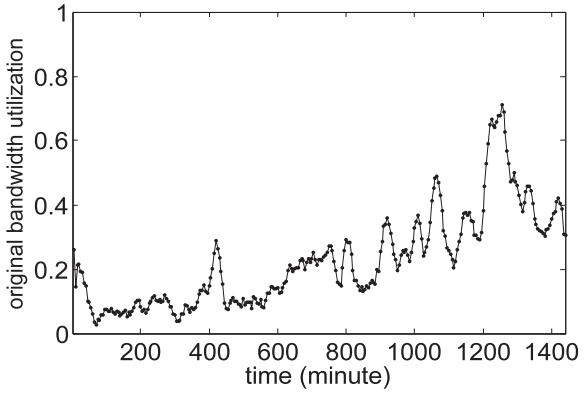


Fig. 6. OBU of the measured servers in 24 hours.

of involved links, novel links, dead links, and inconsistent links in 20 days in December 2011 in Fig. 5. The major observation is that the ratios of novel links, dead links, and inconsistent links (over the involved links) are quite stable around 27, 3.3, and 0.6 percent, respectively. Although the ratio of inconsistent links (0.6 percent) is much smaller than that of dead links (3.3 percent), the negative effect brought by inconsistent links is much larger—for a dead link QQXuanfeng only needs to discard it in the content index DB, while for an inconsistent link QQXuanfeng needs to download the whole linked file and notify the reporting user to restart his download task.

Besides, the link popularity distribution of indexed contents and the user access pattern are presented in Section 3 of the online supplementary file. Moreover, Section 5.2 of the online supplementary file discusses about how to handle the peer swarm dynamics.

4.2 Limited Utilization of Server Bandwidth

Presently, QQXuanfeng tracks over 1 million servers every day. By intelligently directing peer swarms to retrieve data from appropriate servers, QQXuanfeng achieves limited and balanced utilization of the server bandwidth, so that the original services offered by the involved servers are not interfered. As to each server S, by collecting and analyzing users' reports, QQXuanfeng can get its *approximate maximum bandwidth* and its *extra traffic volume* directed by QQXuanfeng. Due to the high dynamics of the Internet, getting the accurate maximum bandwidth of a server is extremely difficult; instead, we adopt a simple *approximate estimation method* as follows. When the content crawler discovers a novel server S, in the subsequent day the data scheduler does not limit the server bandwidth utilization of S but directs as many clients to retrieve data from S as possible. Then the peak upload bandwidth of S during this day is approximately taken as its maximum bandwidth. Besides, even the real maximum bandwidth of S can be changed by its administrator, so the above-mentioned estimation method is periodically performed to update the maximum bandwidth information of S (the update period is usually set as one month).

The worst thing is that QQXuanfeng cannot obtain the *original traffic volume* of a server consumed to support its original service. To get a basic understanding, we obtain the original traffic information of a sample of servers (named "friendly servers") via manual approaches, that is, by directly obtaining the original traffic information of 619 friendly servers (including both file download servers and video streaming servers) from their administrators. Fig. 6 plots the average *OBU* of these friendly servers in 24 hours, starting from 0:00, GTM+8, on December 15, 2011. The curve illustrates an obvious diurnal pattern that accords with people's regular pattern of accessing Internet. The *OBU* of these servers usually stays below *60 percent* and the whole average utilization is merely 22 percent. Furthermore, the measurements done by Heller et al. [23] on 292 servers hosting an e-commerce application and a production Google data center have also revealed low bandwidth utilization of servers (less than 25 percent in average), which is basically consistent with our findings.

Therefore, as to each involved server, the *EBU* directed by QQXuanfeng had better be controlled within *40 percent* ($= 1 - 60$ *percent*). The data scheduler periodically collects users' reports to calculate the *EBU* of each server (the period is 5 minutes). If the *EBU* of a server S exceeds 40 percent, the data scheduler notifies a part of the users served by S to stop their data download from S. For example, suppose the current *EBU* of S is 50 percent and S is uploading data to 100 users. Then the data scheduler notifies 20 users ($20 = 100 \times \frac{50\%-40\%}{50\%}$) who have the highest download rates among the 100 users to stop their data download from S.

Besides, the data scheduler also *shuffles* the server link list to balance the load among involved servers. As to a file f, the data scheduler holds a "link list" of f, denoted as l_f which is returned by the content index DB (see Arrow 5 in Fig. 4). Whenever the data scheduler decides to allocate a partial link list containing n server links to a user, the data scheduler first *shuffles* l_f and then allocates the first n server links in l_f to the user (see Arrow 6 in Fig. 4). The number n will be elaborated in Section 4 of the online supplementary file). Moreover, the user can ask the data scheduler to update his allocated partial link list if his download rate stays below his basic expectation.

4.3 Differentiated Acceleration of Peer Swarms

Although every user of QQXuanfeng hopes to achieve his best experience, through comprehensive measurements and analysis (in Section 4 of the online supplementary file) we discover that a user has his *basic expectation* ($d_{basic} = 30$ KBps) for download rate. Thus, our heuristic is to provide differentiated acceleration of peer swarms.

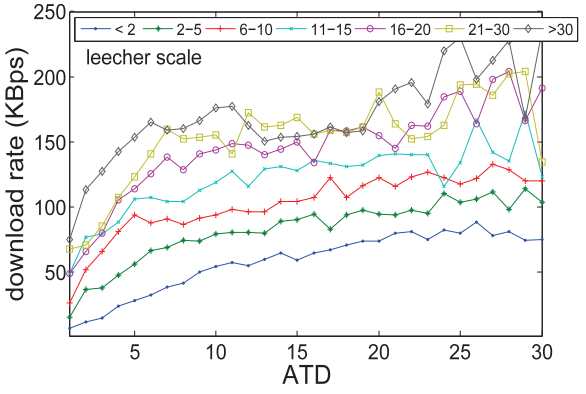


Fig. 7. Relationship between *ATD* and download rate with various leecher scales. ("Leecher scale" denotes the number of online leechers inside a peer swarm.)

Specifically, peer swarms are classified into three categories according to their real-time download rates and data supply demand conditions: 1) *hungry swarms*, 2) *potentially hungry swarms*, and 3) *high-demand swarms*. Different categories of peer swarms correspond to differentiated acceleration strategies so that each user can have a download rate at least above his basic expectation.

- First, a peer swarm is taken as a *hungry swarm* if the average download rate of its peers is below a common user's basic expectation ($d_{basic} = 30$ KBps).
- Second, a peer swarm is taken as a *potentially hungry swarm* if the average download rate of its peers exceeds 30 KBps but its *ATD* is quite low. Since *ATD* is an empirical indicator that illustrates the statistically data supply demand condition of a swarm, low *ATD* often (but not always) implies a *potential* risk of being hungry. Fig. 7 plots the statistical relationship between *ATD* and download rate with various leecher scales (Fig. 7 is obtained from the measurements of 1 million swarms). Accordingly, the setting of *ATD* varies with the leecher scale, as listed in Table 1. Thereby, we use the "ATD of 30 KBps" to recognize potentially hungry swarms.
- Third, a *high-demand swarm* should contain more than ten peers and its file type should be a video type like .rmvb, .mp4, and so on. If a swarm is sharing a video file, its peers may well demand a high download rate ($d_{high} = 100$ KBps) for continuous playback; and the swarm scale is also demanded to be large enough (typically >10) so that the invested server bandwidth is cost effective.[3]

A *hungry swarm* is allocated with a "partial link list" that contains a certain number (n) of server links to enhance its *ATD* to the "ATD of 30 KBps" (see Table 1). For example, if a hungry swarm W_h contains eight leechers and six seed peers sharing a file f and the data scheduler holds a "link list" of f, the data scheduler will allocate $n = 2 \times 8 - 6 = 10$ server links to each peer inside W_h for download acceleration. Here "2" is got from Table 1: Row 3 and Column 4. Note that different peers inside W_h are usually allocated with different partial link lists because the data scheduler always *shuffles* the link list before allocating a partial link list to a peer. Because *ATD* is a statistical indicator rather than an accurate metric, it is possible that some peers still have a download rate below 30 KBps after utilizing the allocated partial link list. Consequently, if the accelerated download rate is still below 30 KBps for a peer p, p will ask the data scheduler to update his allocated partial link list.

For a *potentially hungry swarm* W_p, the allocation strategy of server links is alike except that the allocated server links cannot be used to *really* download data (namely, the allocated server links are just taken as contingency data sources). If the download rate of a peer p inside W_p falls below 30 KBps, p is allowed to really download data from his allocated server links. Similarly, a *high-demand swarm* is allocated with sufficient server links to enhance its *ATD* to the "ATD of 100 KBps."

4.4 Bringing Extra Benefit to Server Providers

QQXuanfeng aims to benefit all the relevant parties: server providers, peer swarms and QQXuanfeng in itself, because any party's resents can hinder or even damage the whole open-P2SP system. Peer swarms get enhanced download rates from QQXuanfeng, but server providers may be reluctant to support QQXuanfeng if they cannot obtain extra benefit from extra bandwidth cost. The two most important benefit metrics of a server provider are its *PV* [14] and *PTC* [15]. Normally, if an Internet user wants to get a file f from a web site (server provider), he needs to click on several links on multiple pages to get the download link, and thus the *PV* of the web site is added by a certain integer (≥ 1). Meanwhile, the *PTC* of the web site is likely to increase if the user clicks on some advertisements on the web pages. However, QQXuanfeng directly allocates the download link of f to the user (of course this brings convenience to the user), so the server provider gets no *PV* or *PTC* additions and loses his economic benefit.

At present, we are fostering benefit collaborations with server providers from both perspectives of *PV* and *PTC*, to encourage more server providers to support QQXuanfeng. Establishing the benefit collaboration scheme involves complicated technical and economic factors, and in fact there exists no commercially mature precedent for us to follow, so our following scheme may be incomprehensive and transitional. On one hand, if a server provider uploads a file f to a peer swarm by the direction of QQXuanfeng, the corresponding *PV* of the server provider will be increased in the web search engine ("Soso" [24]) of QQXuanfeng. Soso is the fourth biggest web search engine in China [25], just lagging behind Baidu, Google China, and Sogou [26]. If a server provider contributes a lot to QQXuanfeng, its corresponding pages or file links will be highly ranked in the search results of Soso, so that the bandwidth contribution of the server provider gets rewarded. On the other hand, QQXuanfeng shares its *PTC* revenue with server providers. QQXuanfeng acquires *PTC* revenue by embedding advertisements into its client software. If a server provider contributes a lot to QQXuanfeng, it will be rewarded by a nontrivial share of our *PTC* revenue.

3. Inside a swarm, the data exchange between its peers is able to "amplify" the invested server bandwidth, i.e., *the amplification factor* $= \frac{\text{peers' download rate increase}}{\text{invested server bandwidth}} > 1$. Generally speaking, the larger the swarm scale is, the larger the amplification factor will be.

TABLE 1
ATD Settings (Corresponding to Fig. 7)

Leecher scale	< 2	2 - 5	6 - 10	11 - 15	16 - 20	21 - 30	>30
ATD of 100 KBps (d_{high})	26	19	11	5	4	4	2
ATD of 30 KBps (d_{basic})	6	2	2	1	1	1	1

One thing worth noting is that the above *PV* and *PTC* methods may take different roles for different open-P2SP systems. For example, as for QQXuanfeng, the *PV* method is the dominant because Soso is a mainstream web search engine in China. But as for Xunlei, the *PTC* method is the dominant because the web search engine ("Gougou" [27]) of Xunlei is relatively weak.

5 Performance Evaluation

This section evaluates the performance of our QQXuanfeng open-P2SP system via comprehensive real-world measurements, as well as specially operated localized and sampling measurements. First, we measure its acceleration effect on peer swarms and how its acceleration effect behaves as the system scales. Then we measure the bandwidth contribution of the involved servers corresponding to different file sizes and swarm scales. Finally, we measure the EBU of the involved servers. The major performance metrics are as follows:

1. *Acceleration effect on peer swarms* represents to what extent the download rates of peers have been increased by QQXuanfeng. This is the kernel performance metric and we extensively evaluate it in three aspects: a) *download rate distribution*, b) *acceleration effect brought by QQXuanfeng*, and c) *acceleration effect as the system scales*.
2. *Bandwidth contribution of servers* denotes the ratio of the users' download bandwidth obtained from third-party servers.
3. *EBU of servers* illustrates the extra bandwidth burden posed on the involved servers by QQXuanfeng.

5.1 Acceleration Effect on Peer Swarms

5.1.1 Download Rate Distribution

Fig. 8 depicts the distribution of peers' download rates in one day (December 15, 2011), where "50" denotes the region [40, 50) and "500+" denotes the region $[400, +\infty)$.

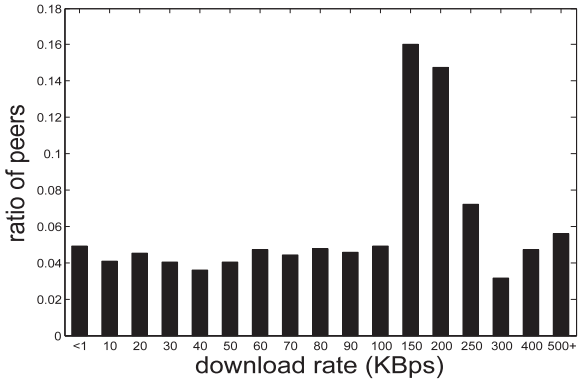


Fig. 8 Distribution of peers' download rates in one day.

Obviously, three download rate regions: [100 K, 150 K), [150 K, 200 K), and [200 K, 250 K) possess the most peers: 16 percent + 15 percent + 7 percent = 38 percent, and more than half (51.4 percent) of the peers have a download rate above 100 KBps (d_{high}). On the other hand, 17.5 percent of the peers have a download rate below 30 KBps (d_{basic}). That is to say, more than 1/6 of the peer swarms are still hungry, mainly because QQXuanfeng does not find sufficient servers that provide the corresponding contents—to deal with this case, we have proposed and implemented a novel *"cloud download"* scheme [28] and have started to implement an ISP-friendly cache mechanism in some cooperative ISPs (refer to Section 5.3 of the online supplementary file). Generally speaking, most users have a download rate above their basic expectations and half of the users have a high enough download rate for continuous video playback.

5.1.2 Acceleration Effect Brought by QQXuanfeng

For a peer swarm W, to evaluate the acceleration effect on its download rate brought by QQXuanfeng, we need to stop the acceleration support to W to measure the *original* download rate of W. Specifically, stopping the acceleration support to a swarm means that the data scheduler does not provide any server link to the peer swarm, and thus the swarm cannot retrieve data from servers. It should be noted that we can only stop the acceleration support to a small number of swarms for a limited period of time to avoid severe degradation in user experience. Therefore, we stop the acceleration support to a random sample of peer swarms (around 1,000 swarms in total) for 7 days, and then recover the acceleration support. The acceleration effect on the download rate is plotted in Fig. 9, indicating that the average download rate of a peer is enhanced from 57 to 158 KBps (177 percent increase). Besides, we record the acceleration effect on the ratio of hungry swarms among the sampling swarms in Fig. 10. The ratio of hungry swarms has fallen from 41.6 to 17.2 percent (58 percent reduction).

5.1.3 Acceleration Effect as the System Scales

As time goes on, QQXuanfeng collects more content links from more servers and meanwhile serves more peer

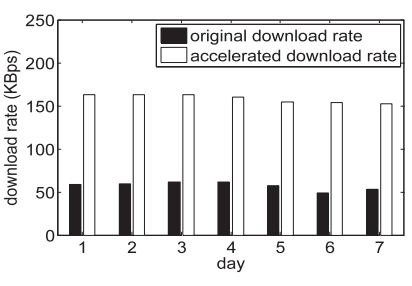


Fig. 9. Acceleration effect on the download rate.

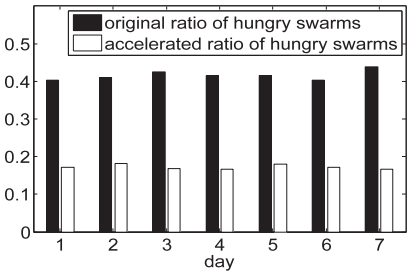


Fig. 10. Acceleration effect on the ratio of hungry swarms.

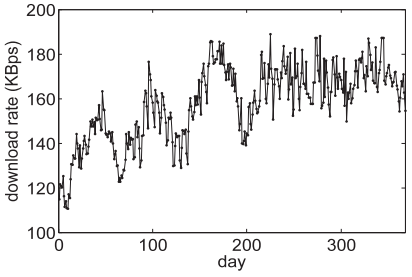


Fig. 11. Average download rate of all peers per day in one year.

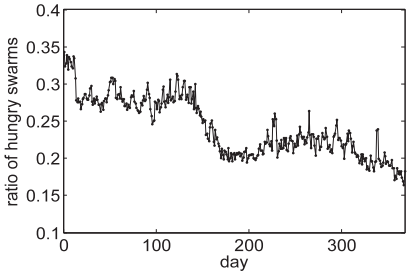


Fig. 12. Ratio of hungry swarms per day in one year.

swarms. For example, in the year 2010, QQXuanfeng gained around 5,000 new users and indexed around 0.6 million novel content links per day. To illustrate how the acceleration effect behaves as the system scales, we record the average download rate of all the peers per day in the whole year, as depicted in Fig. 11. We can see that the average download rate has been enhanced from 115 to 165 KBps (43.5 percent increase). Besides, Fig. 12 presents that the ratio of hungry swarms has fallen from around 34 percent to around 17 percent (50 percent reduction). During the whole year 2010, our hardware architecture generally worked well without adding or upgrading any machine. In conclusion, the acceleration effect of QQXuanfeng tends to be enhanced as the system scales (and some hardware component may need upgrading accordingly).

5.2 Bandwidth Contribution of Servers

A key design target of QQXuanfeng is to let third-party servers contribute their available bandwidth to the peer swarms on demand. In this section, we measure the bandwidth contribution (to peer swarms) of the involved servers. Figs. 13 and 14 plot the server bandwidth contribution corresponding to different file sizes and swarm scales. Generally speaking, when a peer downloads a small file (<100 MB) or when the peer lies in a small swarm (<20), it mainly relies on server bandwidth; on the contrary, when a peer downloads a large file (>1 GB) or when the peer lies in a big swarm (>30), it relies more on its neighboring peers.

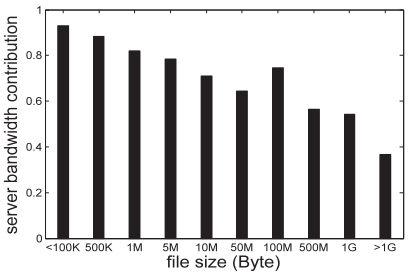


Fig. 13. Server bandwidth contribution corresponding to different file sizes.

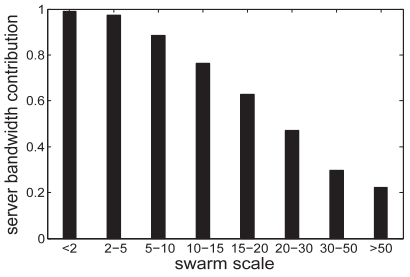


Fig. 14. Server bandwidth contribution corresponding to different swarm scales.

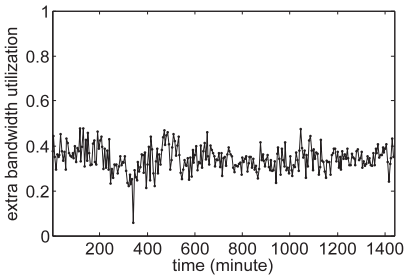


Fig. 15. Average EBU of the involved servers in 24 hours.

The relationship between the server bandwidth contribution and the swarm scale is straightforward. Now, we discuss the impact of file size. Retrieving a small file directly from a server is convenient and quick, while retrieving a small file from peers is not cost effective, since the necessary P2P operations (i.e., establishing and maintaining peer connections) require considerable communication overhead. Moreover, most peers leave the swarm soon after they finish downloading, so it is more difficult to find peers that share a small file. In total, up to 45 percent of the download rate of peers is obtained from the involved servers (note that in Fig. 13, large files have significant impact in computing the total server bandwidth contribution).

5.3 Extra Bandwidth Utilization of Servers

Another important design target of QQXuanfeng is to facilitate load-balancing among third-party servers so that their original services are not interfered. As depicted in Fig. 15, we record the average *EBU* of all the involved servers by the direction of QQXuanfeng per 5 minutes in 24 hours starting from 0:00, GTM+8, on December 15, 2011. In each 5 minute interval, the average *EBU* is mostly below 40 percent. Besides, a "snapshot" distribution of the *EBU* of the involved servers in a five-minute interval is recorded in Fig. 16 (where "0.4" denotes the region [0.3, 0.4]). In this 5 minute interval, the *EBU* of 88 percent

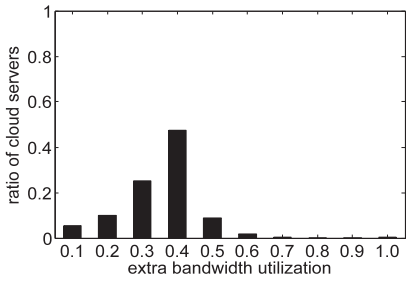


Fig. 16. Distribution of the EBUs of the involved servers in five minutes.

involved servers is below 40 percent. Because QQXuanfeng periodically (the period is 5 minutes) collects users' reports to calculate the *EBU* of each involved server and then take measures to restrict the *EBU* of each involved server, it is reasonable that a small portion of the involved servers are temporarily overloaded. In a word, Figs. 15 and 16 confirm that QQXuanfeng has made limited and balanced utilization of the involved servers.

6 Conclusion and Future Work

Please refer to Section 6 of the online supplementary file for the conclusion and possible future work. This work was performed when Dr. Zhenhua Li was a PhD student at Peking University, Beijing, China.

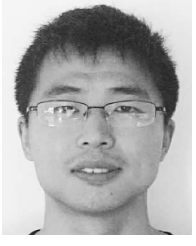
References

[1] QQXuanfeng Web Site, http://xf.qq.com, 2013.
[2] Spotify Web Site, http://www.spotify.com, 2013.
[3] iKu P2P Accelerator, http://c.youku.com/ikuacc, 2013.
[4] UUSee Web Site, http://www.uusee.com, 2013.
[5] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang, "Challenges, Design and Analysis of a Large-Scale P2P-VOD System," *Proc. ACM SIGCOMM,* Aug. 2008.
[6] C. Wu, B. Li, and S. Zhao, "On Dynamic Server Provisioning in Multichannel P2P Live Streaming," *IEEE/ACM Trans. Networking,* vol. 19, no. 5 pp. 1317-1330, Oct. 2011.
[7] Y. Sun, F. Liu, B. Li, B. Li, and X. Zhang, "FS2You: Peer-Assisted Semi-Persistent Online Storage at a Large Scale," *Proc. IEEE INFOCOM,* Apr. 2009.
[8] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky," *Proc. ACM 17th ACM Int'l Conf. Multimedia (MM),* Oct. 2009.
[9] Akamai NetSession Interface. http://www.akamai.com/client, 2013.
[10] Z. Li, T. Zhang, Y. Huang, Z.L. Zhang, and Y. Dai, "Maximizing the Bandwidth Multiplier Effect for Hybrid Cloud-P2P Content Distribution," *Proc. IEEE/ACM 20th Int'l Workshop Quality of Service (IWQoS),* June 2012.
[11] Xunlei Web Site, http://www.xunlei.com, 2013.
[12] Flashget Web Site, http://www.flashget.com, 2013.
[13] Orbit Web Site, http://www.orbitdownloader.com, 2013.
[14] PV (Page View), http://en.wikipedia.org/wiki/Page_view, 2013.
[15] PTC (Paid-to-Click), http://en.wikipedia.org/wiki/Paid_To_Click, 2013.
[16] Rayfile Web Site, http://www.rayfile.com, 2013.
[17] ChinaCache CDN, http://www.chinacache.com, 2013.
[18] P. Aditya, M. Zhao, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, and B. Wishon, "Reliable Client Accounting for P2P-Infrastructure Hybrids," *Proc. Ninth USENIX Conf. Networked Systems Design and Implementation (NSDI),* Apr. 2012.
[19] P. Dhungel, K. Ross, M. Steiner, Y. Tian, and X. Hei, "Xunlei: Peer-Assisted Download Acceleration on a Massive Scale," *Proc. 13th Passive and Active Measurement Conf. (PAM),* Mar. 2012.
[20] M. Zhang, W. John, and C. Chen, "Architecture and Download Behavior of Xunlei: A Measurement-Based Study," *Proc. Second Int'l Conf. Education Technology and Computer (ICETC),* June 2010.
[21] M. Zhang, W. John, and C. Chen, "A Measurement-Based Study of Xunlei," *Proc. 10th PAM Student Workshop,* Apr. 2009.
[22] http://tech.163.com/11/0728/07/7A1J9GKV000915BF.html, 2013.
[23] B. Heller et al., "ElasticTree: Saving Energy in Data Center Networks," *Proc. USENIX Conf. Networked Systems Design and Implementation (NSDI),* Apr. 2010.
[24] Soso Search Engine, http://www.soso.com, 2013.
[25] A Report on China's Search Engine Ranking (Apr. 2011), http://search.iresearch.cn/32/20110412/136979.shtml, 2013.
[26] Sogou Search Engine, http://www.sogou.com, 2013.
[27] Gougou Search Engine, http://www.gougou.com, 2013.
[28] Y. Huang, Z. Li, G. Liu, and Y. Dai, "Cloud Download: Using Cloud Utilities to Achieve High-Quality Content Distribution for Unpopular Videos," *Proc. 19th ACM Int'l Conf. Multimedia (MM),* Nov./Dec. 2011.

Zhenhua Li received the PhD degree in computer science and technology from Peking University, Beijing, China. Then he joined the School of Software and TNLIST, Tsinghua University, Beijing, China. His current research areas mainly consist of Internet content distribution, cloud storage and P2P (peer-to-peer) technologies. He has published one book and 27 technical papers in the above areas. He is a member of the ACM, ACM SIGMM, and the China Computer Federation.

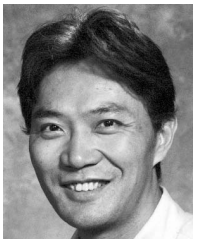
Yan Huang received the bachelor's and master's degrees both from the University of Science and Technology of China. He is currently working at the Baidu Antivirus, Shanghai, China. As a director, he used to lead the P2P and cloud computing related projects of Tencent Research. Before joining the Tencent company, he cofounded PPLive (now known as PPTV.com) and acted as the chief architect. He has published papers in multiple well-recognized conferences like SIGCOMM, ACM-MM, ICCCN, and so forth.

Gang Liu received the bachelor's and master's degrees both from the Huazhong University of Science and Technology, China. As a system architect, he is currently taking charge of the P2P and cloud computing related technologies of Tencent Research, in particular the Tencent cloud downloading and cloud transcoding systems. He has published papers in referred conferences/workshops like ACM-MM and NOSSDAV.



Fuchen Wang received the bachelor's and master's degrees both from the Huazhong University of Science and Technology, China. As a system architect, he used to take charge of the P2P and cloud computing related technologies of Tencent Research, in particular the Tencent cloud downloading and cloud transcoding systems. He is currently working at the Baidu Antivirus, Shanghai, China.

Yunhao Liu received the BS degree in automation from Tsinghua University, Beijing, China, in 1995, the MS and PhD degrees in computer science and engineering from the Michigan State University, in 2003 and 2004, respectively. He is now the EMC chair professor at Tsinghua University, as well as a faculty member with the Hong Kong University of Science and Technology. His research interests mainly include wireless sensor network, peer-to-peer computing, and pervasive computing. He is a senior member of the IEEE and the IEEE Computer Society.



Zhi-Li Zhang (M'97-SM'11-F'12) received the BS degree from Nanjing University, Jiangsu, China, in 1986, and the MS and PhD degrees from the University of Massachusetts, Amherst, in 1992 and 1997, respectively, all in computer science. In 1997, he joined the Computer Science and Engineering faculty with the University of Minnesota, Minneapolis, where he is currently a professor. From 1987 to 1990, he conducted research with the Computer Science Department, Aarhus University, Denmark, under a fellowship from the Chinese National Committee for Education. He has held visiting positions with Sprint Advanced Technology Labs, Burlingame, CA; IBM T. J. Watson Research Center, Yorktown Heights, NY; Fujitsu Labs of America, Sunnyvale, CA; Narus Inc., Microsoft Research; INRIA, Sophia-Antipolis, France; Universidad de Carlos III de Madrid and IMDEA Networks. He is a corecipient of three Best Paper Awards from ACM SIGMETRICS, IEEE ICNP, and IEEE INFOCOM. He is a fellow of the IEEE.

Yafei Dai received the PhD degree in computer science and technology at the Harbin Institute of Technology, China. She is a professor at the Department of Computer Science and Technology, Peking University, Beijing, China. Her research areas mainly include networked and distributed systems, P2P computing, network storage and online social networks. She is a member of the IEEE, IEEE Computer Society, and China Computer Federation.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.