Peer-to-Peer Netw Appl
DOI 10.1007/s12083-007-0003-1

On peer-to-peer (P2P) content delivery

Jin Li

Received: 16 May 2007 / Accepted: 29 November 2007

Abstract In both academia and industry, peer-to-peer (P2P) applications have attracted great attentions. P2P applications such as Napster, Gnutella, FastTrack, BitTorrent, Skype and PPLive, have witnessed tremendous success among the end users. Unlike a client-server based system, peers bring with them serving capacity. Therefore, as the demand of a P2P system grows, the capacity of the network grows, too. This enables a P2P application to be cheap to build and superb in scalability. In this paper, we survey the state of the art of the research and the development of P2P content delivery application. Using examples of the deployed P2P applications and research prototypes, we survey the best practices in P2P overlay building and P2P scheduling. We hope that the information may help the readers to build a reliable, robust P2P content delivery application.

Keyword Peer-to-peer (P2P) · P2P file sharing · P2P streaming · P2P broadcast · Survey · Overlay · Scheduling · Efficiency · Reliability · Robustness · Quality of service (QoS)

1 Introduction

In May 1999, an 18-year old college student, Shawn Fanning, built Napster in the dormitory of Boston's Northeastern University [2]. One of Shawn's college roommates loved listening to MP3s and complained about the unreliable MP3 sites these days. Music links on those sites were frequently dead, and indexes were often out of date. In response, Shawn developed a real-time system to help the music lovers to find and exchange music files on the Internet. Unlike the popular approach of the day, which sent out robots to roam the Internet periodically, search, update, and remove music that it found on the Internet, Shawn's program had users listed the files they were willing to share on a computer that others can access. Moreover, users of Shawn's program directly downloaded the music from other users' computer, bypassing the use of central server. Shawn named the program Napster, which was his nickname, email alias, and his username in IRC rooms.

Napster was released in an era of rapid growth of the Internet bandwidth, CPU power and disk storage capacity. Couples with the fact that the Internet access in most American homes and campuses was flat rate, Napster quickly became popular. In Feb. 2001, Napster boasted a peak of 1.5 million simultaneous users [3]. Although the original Napster service was soon shut down by a court order [1], it made a major impact on how people, especially students used the Internet. The traffic analysis conducted by CacheLogic [4], shown in Fig. 1, illustrates that ever since year 2003, P2P file sharing traffic has surpassed web traffic and becomes the single largest traffic type by volume on ISP networks. Now on the Internet backbone, 65–70% of the traffic can be attributed to the P2P. In the last mile, 50–65% of downstream and 75–90% of upstream traffic is P2P traffic. Moreover, P2P file sharing traffic continuous to grow.

Napster makes the term peer-to-peer (P2P) popular. However, the concept and technology behind P2P may find its origin in a number of much older technologies, such as IP routers, DNS (domain name service), Usenet news server system, FidoNet [7], etc., in which the technology trend moves away from monolithic systems and toward

J. Li (✉)
Microsoft Research, Communication and Collaboration Systems,
One Microsoft Way,
Redmond, WA 98052, USA
e-mail: jinl@microsoft.com

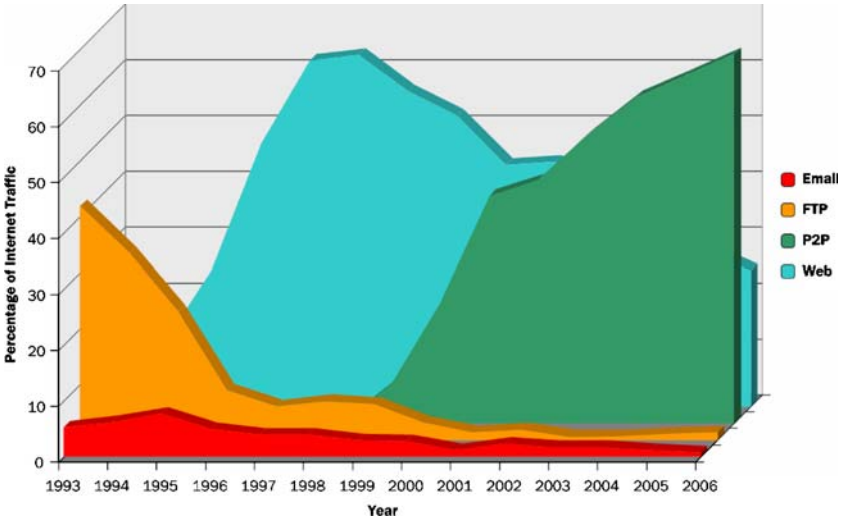

Fig. 1 Internet protocol trend, 1993 to 2006 (source: Cache Logic [4])

distributed systems. The Usenet can be considered an earlier implementation of P2P content delivery. The Usenet server communicates in P2P fashion among each other to exchange Usenet news article over the entire network. Each individual user of Usenet then communicates in a client-server fashion with the local Usenet news server to read and post articles.

With Napster and the following P2P applications leading a revolution in file sharing and other type of content delivery, it is interesting to clarify what is exactly P2P. In this paper, we take a resource centric approach to classify P2P. Following the proposal of Clay Shirkey [5], P2P is defined as *a class of applications that take advantage of the resources – storage, cycles, content, human presence – available at the edges of the Internet*. Compared with the traditional client-server architecture of the Internet, a P2P application has the important properties that each peer node belongs to a different owner, and pays and contributes its own resource in return of the service rendered by the P2P network. Thus, the peer is both a resource provider and a resource consumer. As more and more nodes join the P2P network and demand on the system increases, the total resource of the system also increases. Thus, a P2P application is economy to run, robust and super-scalable. This is in sharp contrast to the cost structure of the client-server application, where it is the server that is paying for the capacity. Moreover, since each peer in the P2P network is a service provider, the servicing resource of the P2P network is distributed with redundancy, this leads to a robust service. Properly designed, P2P network may also efficiently utilize the geographical and ISP diversity of the peers to smartly route the content from the source to the peer nodes, and reduce the traffic on the backbone of the Internet.

Schollmeier [6] further defines a *pure* P2P network as the one that any single, arbitrary chosen peer can be removed from the network without having the network suffering any loss of network service. In comparison, a *hybrid* P2P network has a central entity (server) which renders certain central functionality of the service, e.g., keeps track of peers and responds to the request of peers. Either pure and hybrid P2P network differs from the traditional client-server application in that the peers contribute the majority of the resource running the service, which includes the CPU power, the bandwidth and the storage.

We can also classify the P2P network based on how the content is delivered and consumed. For simplicity, let us assume that the source file to be distributed is a 400kbps video file created on the fly, and use it as an example to illustrate various content delivery method. The most unrestrictive mode of delivery is bulk download. Most popular P2P file sharing utilities, such as BitTorrent, eDonkey, FastTrack and Gnutella belongs to this mode. In the bulk download mode, the peer doesn't care about the source characteristics (video file at 400 kbps). Its operation goal is simply to retrieve the file from the P2P network as fast as possible. For efficiency, most modern P2P file sharing applications, such as BitTorrent or eDonkey, chop the file into blocks (also called pieces, packets) and deliver the chopped block in non sequential order. Therefore, the shared video is not playable until the entire video is retrieved. But even on a slow connection, the video will look great during playback. The bulk download guarantees content quality with unbounded delivery time. As such, when the actual network bandwidth is tight, say 33.6 kbps modem, you may have to wait a long time, hours or days, before you can play the file.

Since a large amount of content delivered over the Internet is video or music, where the user favors fast response, an alternative mode of content delivery, *streaming*, is born. In the streaming mode, the application is aware

of the format, the required bandwidth, and the structure of the delivered content, and allows the content to be played smoothly during the delivery. This involves a variety of technologies. For one, the streaming application usually uses a playback buffer at the client. At the beginning of the streaming, the playback buffer is filled with the arriving content while the playback is suspended. The playback only starts when the playback buffer reaches a certain level. During the playback, the streaming application carefully manages the buffer and ensures a certain buffer occupancy level, so that the playback will not be interrupted due to packet loss or jitter during the delivery, and the buffer is not excessively long which consumes unnecessary server bandwidth. The streaming application may also adapt the delivery media according to the resource available in the system and to the client. For example, for client at the end of a slow Internet link or for P2P network with insufficient total network bandwidth, a low bit rate version of the video can be streamed to the client, if the video is encoded with multiple bit rate (MBR) streams [8] or scalable stream. Again using the 400 kbps video as an example, in the streaming mode, the video may be transcoded, either through a scalable video codec or through a MBR video codec to different bit rates, say 200 kbps, 100 kbps, 50 kbps and 25 kbps. For client at the end of a 33.6 kbps modem link, we can opt to deliver only the 25 kbps video stream. The streaming client may also opt to drop packets that have a less impact on playback quality, e.g., a B frame video packet, to ensure that the video can still be smoothly played back using the available resource. Overall in streaming, the target is to get a smooth playback experience and a bounded delivery time, which is determined by the playback buffer. However, the content quality may be sacrificed. The delivery time delay depends on the fluctuation of the available resource to the client and in the P2P network. In practical P2P streaming implementations, such as PPLive, the delivery time delay can be 15–120s.

A third mode of content delivery requires bounded delay from the creation to the consumption (the content is played back at the client). We call the delivery mode as *delay bounded broadcast*, or simply *broadcast*. Broadcast mode applications include: audio/video conferencing, interactive gambling, earning conference call (or distribution of other financial information), Internet gaming, etc. Compared with the streaming application, the broadcast application not only needs to sustain playback during delivery, but also needs to put an upper bound on the maximum delay. As a result, the content delivery algorithm has to be optimized for minimizing delay, and certain tricks in the streaming application, e.g., the use of the playback buffer to combat packet loss and jitter, can not be used. There are many research prototypes for P2P broadcast applications, such as Microsoft ConferenceXP [63], NICE [30], Zigzag [31], ESM [63]. However, compared with P2P file sharing and P2P streaming, the current deployment scale of P2P broadcast is much smaller in the real world.

Whether it is P2P file sharing, P2P streaming, P2P broadcast, a well designed P2P application should have the following characteristics. It is efficient, meaning that it uses all resource available in the P2P network. It is robust, i.e., it can cope effectively with the changes and the anomaly of the peers and the network links. It satisfies the quality of service (QoS) requirement of the delivery mode. With streaming, this means that the playback is smooth with as high media quality as possible. With broadcast, this means that it satisfy certain delay constraint of the delivery of the media.

In this paper, we survey the state of the art of the development of efficient and robust P2P content delivery applications. The paper is organized as follows. We will start by examining a number of widely deployed P2P applications in Section 2. We then examine the two important aspects to achieve the efficiency and robustness: the P2P overlay construction (in Section 3) and the P2P scheduling algorithm (in Section 4). We use both widely deployed P2P applications, such as Gnutella, eDonkey, FastTrack, BitTorrent, PPLive, and Skype, and highly cited P2P research prototypes to illustrate the design strategies adopted by developers and researchers. We summarize the current state of the P2P delivery technologies and propose future works in Section 5.

2 Anatomy of widely deployed P2P applications

In this section, we examine several P2P applications. Between widely deployed P2P applications with large user base and widely cited P2P research prototypes, our discussion favors the former. We will delay the discussion of the latter when we examine the components of P2P overlay construction and P2P scheduling in Section 3 and 4. The purpose of the section is to use the real world P2P application as guidance to steer the development of future P2P applications.

2.1 P2P file sharing applications

After Napster raised the interest of P2P, a great number of P2P file sharing applications have been developed all over the world with different design philosophies and operation modes. The most popular P2P file sharing applications are Gnutella, FastTrack, eDonkey and BitTorrent.

a) Gnutella

Gnutella, developed by Justin Frankel and Tom Pepper of Nullsoft in 2000, is one of the earliest P2P file sharing tools following the Napster. Gnutella works as follows [9].

Upon initial connection, a Gnutella peer joins the network via at least one known peer, whose IP address is obtained either via an existing list of pre-configured addresses, or via a bootstrapping process that talked to a known source of the Gnutella peer, e.g., a web server, a UDP host, or even IRC. Through this initial Gnutella peer, the new coming peer will discover more new Gnutella peers until the number of its direct neighbors reaches a predetermined threshold. When the user wants to do a search, the source peer will send the search request to all actively connected peers. The recipient peer answers the query if it knows anything useful, or forwards the request. The query thus propagates among the Gnutella network until a predetermined number of hops is reached. Essentially, Gnutella floods the network to conduct the search. If a peer has the requested file of the search, it will contact the source peer. If the user of the source peer decides to download the file, it will negotiate the file download with the target peer.

Gnutella is one of those pure P2P applications that do not have a centralized server. A noticeable feature is that Gnutella is under nobody's specific control and is effectively impossible to shutdown. The purely decentralized Gnutella is made popular by Napster's legal trouble in early 2001, in which Judge Madelyn Patel ruled that since Napster used servers that indexed and routed copyright-infringing music sharing, it was knowledgeable of the illegal file sharing activity of its users and was thus liable. However, the growing surge in popularity reveals that the initial Gnutella protocol has limitation in scalability. The number of Gnutella's search requests grows exponentially to the number of connected users [10, 11]. When the number of Gnutella users grows, the search requests overwhelm the Internet, and thus get prematurely dropped. As a result, the search only reaches a small portion of the network. To address the problems, later version of Gnutella implements a two class system of supernodes and leaves, with only the supernodes perform the search and routing functionality.

b) FastTrack

FastTrack was introduced in March 2001 by Niklas Zennström, Janus Friis and Jaan Tallinn. Three popular P2P clients use FastTrack - Kazaa, Grokster and iMesh. A noteworthy feature of FastTrack is that it uses supernodes to improve scalability. A FastTrack peer with powerful CPU and fast network connection is automatically promoted to become a supernode, which acts as an index server for other slower FastTrack peers. FastTrack also employs UUHash to find identical files in multiple peers so that it can download the file simultaneously from multiple sources. Unfortunately, though the UUHash is simple to implement and fast to compute, it is rather weak, leading FastTrack susceptible to pollution attacks [12].

c) eDonkey

The eDonkey was conceived in Sept. 2000 by MetaMachine Inc. The eDonkey network operates as a hybrid P2P network similar to Napster [13]. The eDonkey network consists of clients and servers. The eDonkey server acts as a communication hub, indexes files and distributes addresses of other eDonkey servers to the clients. The eDonkey client downloads and shares the files. The eDonkey network is based upon an open protocol. As a result, there are dozen versions of eDonkey servers and clients. In the earlier days, the most popular eDonkey client was the eDonkey2000. However, since 2005, eMule, a free software implementation of eDonkey client that was started on Mar. 2002 by Hendrik Breikreuz has taken over and becomes the most popular one.

In the earlier days, eDonkey relies on users to run servers on a volunteer basis to keep the network operating. The small cluster of eDonkey servers often overloads, and is vulnerable to attacks. To overcome the problem, both eDonkey2000 and eMule have resorted to the Kademlia DHT [38] (distributed hash table) algorithm to distribute the load of the server. With the Kademlia DHT, eDonkey 2000 becomes Overnet, while eMule becomes the Kad network. This solves the scaling problem.

d) BitTorrent

BitTorrent was created in 2002 by Bram Cohen. It runs on an open protocol. The original BitTorrent client was written in Python. But today, there are more than a dozen compatible clients written in a variety of computer programming languages [14]. To share a file or a set of files through BitTorrent, a torrent file is first created. The torrent file contains the metadata of the shared content, which includes the information of the tracker that coordinates the file distribution, and the hashes of the file blocks to be distributed. BitTorrent protocol doesn't specify how the torrent file is distributed. In fact, the torrent file is usually distributed by traditional means, e.g., posting on the web, through a forum, BBS board, or through the search engine run by specific BitTorrent society. When a BitTorrent client wants to retrieve a file, it first obtains the torrent file. The client then contacts the tracker listed in the torrent file, which can be a single computer or a distributed set of computers in the trackerless BitTorrent implementation, to obtain a list of peers that are sharing the file at the same time. BitTorrent adopts a Tit-for-Tat content sharing strategy. At a certain time instance, a BitTorrent peer preferentially uploads to top m (default $m=5$) neighbors that provide the peer with the best download rate. To jumpstart sharing and to make sure that a new arrival BitTorrent peer may contribute its resource, BitTorrent also uses an optimistic unchoking strategy that violates strict Tit-for-Tat. The optimistic unchoking allows the peer to upload to a random neighbor peer regardless of its download rate. BitTorrent also adopts a local rarest content distribution rule. That is, a BitTorrent peer will first download the block that is the rarest in its local neighbor-

hood. This ensures that with good probability, the newly downloaded block can be further distributed to the other peers, thus efficiently utilizing the upload bandwidth of the current peer.

A number of unique features have allowed BitTorrent, a relative late comer in P2P file sharing, to become a popular file sharing utility. First, BitTorrent itself does not offer any index or search mechanism to find sharing files, it thus distances itself from copyright infringement accusation. Second, the Tit-for-Tat content sharing mechanism, though simple, is a remarkable effective mechanism to encourage BitTorrent users to share and contribute. As a result, BitTorrent suffers far less from the leech or the free-rider problems that plaque the other P2P file sharing applications. Third, by recording hashes of sharing blocks in the torrent file, BitTorrent avoids the pollution attack [12] troubling prior file sharing applications. Fourth, the action to split file into sharing blocks and to share blocks on a local rarest basis is a very effective mechanism to utilize the bandwidth resource of the peer and to achieve high efficiency. In fact, BitTorrent can be considered the first widely deployed P2P file sharing application that achieves both efficiency and robustness.

BitTorrent has generated great enthusiasm for P2P file distribution in academia and industry. Many open/free source software projects, e.g., OpenOffice.org, SUSE and Ubuntu Linux distribution, have used BitTorrent to distribute their source code and compiled binary. World of Warcraft, a popular multiplayer online role-playing game (MMORPG), uses BitTorrent to distribute game patches. Warner Brothers Entertainment plans to distribute the films and TV shows over the Internet using BitTorrent. Fan-film producers and small music makers have made their file, music, footage available via BitTorrent. A number of conferences have distributed their conference video recording through BitTorrent as well. Many small business content owners comment that without BitTorrent, they simply can not distribute content in a cost effective yet efficient fashion.

e) P2P file sharing statistics

According to CacheLogic [4], BitTorrent, eDonkey, FastTrack and Gnutella are the four most popular P2P file sharing utilities used today. In Fig. 2, we show the percentage share of P2P traffic in different countries of the four file sharing utilities. In general, BitTorrent and eDonkey are by far the most popular file sharing tools, the FastTrack (Kazaa, Grokster and iMesh) is a distant third, and Gnutella takes a fourth position. The popularity of the file sharing utilities is also very different in countries. For example, in South Korea, eDonkey is much popular than BitTorrent; while in Singapore, the reverse is true; and in United States and China, eDonkey and BitTorrent are equally popular.

2.2 P2P voIP-skype

Let's take a look at Skype, a P2P VoIP (voice over Internet protocol) application. Skype was released on Aug. 2003 by Niklas Zennström and Janus Friis, who founded FastTrack and Kazaa. Compared with public VoIP standard such as SIP and H.263, Skype uses a proprietary protocol and relies on the Skype P2P network for user directory and firewall/NAT (network address translator) traversal. As a result, Skype can bypass the costly infrastructure associated with the deployment of VoIP, and can easily scale to a very large size.

Like FastTrack built by the Skype founders, Skype uses a two tier infrastructure, which consists of supernodes and clients. When a peer joins the Skype network, it always

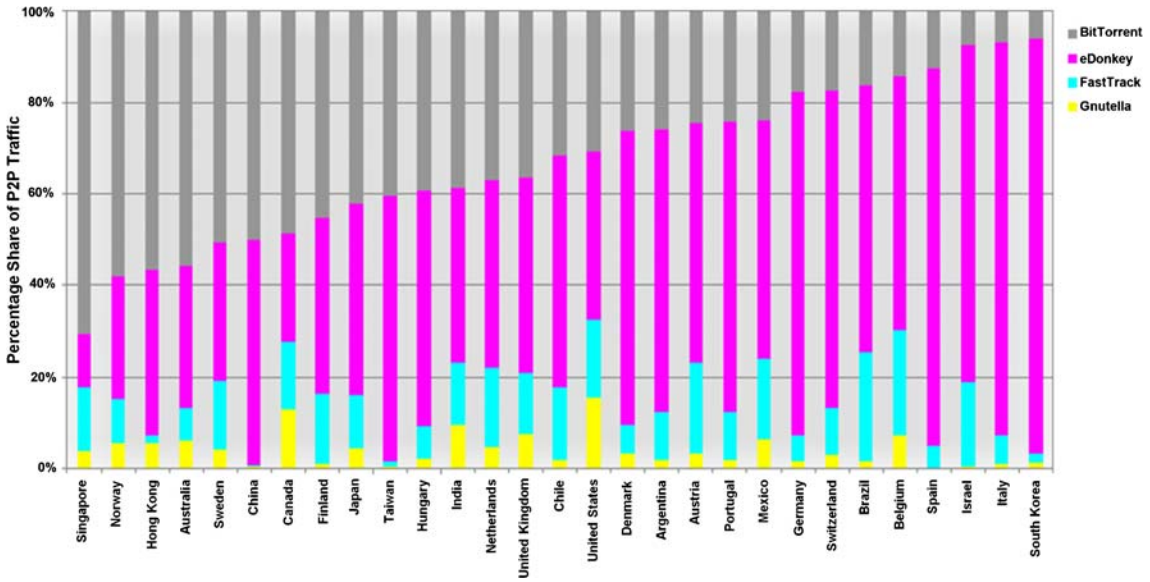

Fig. 2 Percentage share of P2P Traffic (BitTorrent, eDonkey, FastTrack and Gnutella, source: CacheLogic [4])

starts as a client node. Then the Skype software checks if the client has sufficient CPU, memory and network bandwidth, and if it is directly connected to the Internet (not behind any NAT or firewalls). If all the above conditions are satisfied, Skype automatically promotes the client to the supernode. Once becoming a supernode, the Skype peer will performs user directory service to help other Skype users, and to relay traffic for computers behind firewall/NAT. The Skype client follows a set of rules to detect the existence of firewall/NAT, and to ensure that a Skype connection, either directly or indirectly, can always be established between two Skype peers. This is performed as follows. The Skype client always finds at least one Skype supernode as its default gateway, and first tries to establish a UDP connection to the supernode. If that fails, it tries to establish a TCP connection on an arbitrary port to the supernode. If that still fails, it tries to establish the connection to the supernode on port 80 (the Http port) or port 443 (the SSL port). Because most corporate firewalls at least allow the Http and/or Https (SSL) traffic, the fall back mechanism to connect on port 80 and port 443 almost always works. The same fall back to port 80 and/or port 443 mechanism has been widely used in instant messenger client, such as MSN/Live Messenger, Yahoo Messenger, ICQ, etc.. When two Skype users intend to establish a communication session, Skype first tries to establish a direct connection with a certain NAT traversal mechanism. If the direct connection fails, Skype relays the traffic through the gateway supernode of the Skype clients. The use of supernode relay enables Skype connection to work in situation where SIP/H.263 fails, and enable Skype to use a high bit rate wideband speech codec (24 kbps) to achieve good perceptual voice quality while doesn't incur the cost of relaying traffic for peers behind firewall/NAT.

Another key distinction between Skype and SIP/H.263 is that Skype utilizes a strong encryption protocol to protect the confidentiality of the call. Skype certifies its user's public key during login via a 1536-bit (for unpaid Skype account) or a 2048-bit (for paid Skype account) RSA certificate. It uses a 1024-bit RSA to negotiate symmetric AES (advanced encryption standard) key, and uses 256-bit AES to protect calls. The feature of encrypted Skype calls has attracted many corporate and consumer users.

Skype is immensely popular. In October 2006, it boosts 136 million registered users with $50 million revenues. This represents a 20% increase in the number of registered users and a 13% increase in the revenue, compared with the figure of July 2006, just three month ago. The revenue is mainly derived from Skype users who use SkypeIn and SkypeOut service, in which SkypeIn allows Skype users to receive calls on their computer dialed by regular phone subscribers, and SkypeOut allows Skype users to call traditional telephone users. Skype has deployed both PSTN (public switched telephone network) gateway and Skype-to-SIP/H.323 gateway to allow SkypeIn and/or SkypeOut functionality.

Three years after its release, Skype handles 7% of world's long-distance minutes. Moreover, Skype has already gone beyond traditional PC platforms. Its client is available for Microsoft Windows (2000, XP, CE), Mac OS X and Linux. Netgear® and Motorola® has released Wi-Fi Skype phone (shown in Fig. 3a,b). Wherever you have Wi-Fi access, these phones will allow you to calls anyone else on Skype, anywhere in the world for free, without using a PC. And with SkypeIn and/or SkypeOut, you can communicate with ordinary phones worldwide for pennies per minute. Shown in Fig. 3c, IPEVO's USB Skype phone allows the use of the traditional phone interface to call users via Skype. The IPdrum mobile Skype cable, shown in Fig. 3d, allows people with phone contracts that can make free calls to pre-defined numbers (such as "friends & family" plan) to make cheap VoIP calls from their mobile phone to anywhere in the world. The trick is to hook one mobile phone to the IPdrum mobile Skype cable, which is further hooked to a computer running Skype, and make calls use another mobile phone which is on the friends &

Fig. 3 Skype gadgets: (**a**) Netgear Skype Wi-Fi phone, (**b**) Motorola CN620 Skype Wi-Fi phone, (**c**) IPEVO Free-1 USB Skype phone, and (**d**) IPdrum mobile Skype cable

family plan of the stationary mobile phone. The fixed mobile phone serves as a gateway for the Skype application, and allows the user to receive and make mobile phone calls to anyone in the world via SkypeIn and/or SkypeOut.

2.3 P2P streaming

CacheLogic data [4] also shows that the average file size for P2P file sharing is constantly growing, and the majority of P2P traffic volume is generated by objects with an average size greater than 1 GB. This suggests that P2P file sharing is moving strongly towards video/music sharing. The era of P2P streaming, which is a much favorable delivery mode for large media, is coming. The target of P2P streaming is to build a scalable P2P platform for TV/music delivery. More than a dozen companies are actively working in this area. Some example companies are RawFlow [15], Abacast [16], PPLive [17], PPStream [18], UUSee [19], Roxbeam [20], Mysee [21], etc.

Let us use PPLive, arguably the most popular P2P streaming application nowadays, as an example. At the end of 2005, PPLive has 20 million download, and has 1 million independent viewers per day. According to [17], it supports over 200,000 concurrent users at bit rate in the 400–800 kbps for 2006 Spring Festival Gala on Chinese New Year on January 28, 2006. In 2007, the number of concurrent users for the most popular PPLive session raises to 1.5 million [73]. This corresponds to an aggregate bit rate in the vicinity of 600 Gbps, or 540TB transferred per the 2 hour event. During the event, the PPLive server only provides 10 Mbps distribution bandwidth, or 0.0017% of the bandwidth. From the preliminary analysis of [22], PPLive operates as follows. When the PPLive client is launched, it retrieves from a channel server the metadata information of all channels. Currently, PPLive offers approximately 300–400 channels, with all viewers at the same channel watch the media at approximately the same point. The PPLive client presents the channel list to the user, who selects one particular channel to watch. After the channel selection, the PPLive client further talks to a tracker of that channel, and retrieves a list of peers that are watching the same channel. The PPLive client connects to a set of peers, and starts to exchange data. During the exchange, it chops the streaming media to chunks, with each chunk being about 1s worth of compressed media. The PPLive client exchanges a buffer map that is coded chunk availability within a future playback window that is about a few minutes long, and makes decision on the particular chunks to retrieve and/or to send via a proprietary algorithm. At the same time, the retrieved chunks are stored in a buffer in the PPLive engine, and are fed through a local Http pipe to the Windows Media player or the Real player, depending on the type of the media being streamed.

The challenge in P2P streaming is to provide a sustained streaming bit rate to all peers joining the network. Unlike P2P file sharing, where the content delivery can be carried out on a best effort basis, in P2P streaming, insufficient delivery bandwidth leads to poor quality of service, such as playback stalling or freezing, which is very annoying to the users. Through analysis of [22], we learn that the current PPLive platform do experience playback freeze for as long as 1 minute, and as frequent as 4 freezing incidents in a 7 minute interval. The experience of PPLive demonstrates that large scale video streaming can be done. But there is still room for improvement on efficiency and robustness. PPLive also incurs a relatively long playback lag. It is observed that PPLive can incur startup delay for about 20s to 30s for popular channels, and up to 2 minutes delay for unpopular channels, and some peers may watch frames in a channel minutes behind others.

2.4 Lessons learnt from deployed P2P applications

P2P application revolutionizes the cost structure of the Internet application, and makes large scale content delivery with low server cost feasible. The successfully deployed P2P application above shares some common traits. All successful P2P applications greatly reduce the cost of running the source server. Whether the system operates in a pure P2P or in a hybrid P2P mode is inessential. In fact, many P2P applications offer the capability to run the system in both modes. For example, BitTorrent can be run in either tracker mode or the trackerless mode. The eDonkey application can be run either via an eDonkey server or via a Kademlia DHT, which eliminates the server. The end users do not care whether the system is pure P2P per se.

Nearly all the widely deployed P2P applications have improved the quality of service (QoS) from the end user perspective. For many content owners on the long tail that are without deep pocket, the P2P File sharing tools such as BitTorrent remain their only choice to reach a large group of users in a cost effective fashion. Their users are willing to use the P2P application because it greatly improves the speed that the content can be retrieved. By utilizing supernode assisted relay, Skype is able to increases the speech codec bit rate, and offers the user a higher quality of service. Without P2P streaming tools, large scale distribution of TV/video usually has to use a much lower media coding bit rate, renders the resultant compressed video unattractive from the end user's perspective. Often, it is the improvement of QoS that attracts the user to go through the trouble to download and install the related P2P applications.

Most successful P2P applications have certain incentive system to encourage the end users to contribute their resource to the P2P network. This can be a voluntary

incentive or an involuntary incentive. BitTorrent uses a voluntary incentive via the Tit-for-Tat sharing protocol. BitTorrent allows the user to set the upload bandwidth. However, if the user reduces the upload rate, his/her rate of download suffers. This deters the user to become a free rider, who consumes resource but does not contribute. PPLive and Skype adopt an involuntary contribution model. Both applications do not offer the user the choice of whether he/she wants to contribute, and how much to contribute. Rather, the peers with more resource are automatically drafted to the service of the other less resourceful peers. For Skype case, the peers that are not behind NAT, with abundant bandwidth and CPU resource are automatically drafted for the service. For PPLive, the peers that have more upload bandwidth are automatically drafted to serve the other peers. Because Skype and PPLive offer their users a valuable service and good QoS, which for Skype it is the VoIP that reaches anywhere, and for PPLive, it is the opportunity to watch TV program not accessible in its local market, such involuntary contribution is tolerated by their user base so far.

Securities are key concerns in widely deployed P2P applications. The relatively older P2P file sharing applications suffers from a variety of attacks. FastTrack and eDonkey have suffered extensively from pollution and index poisoning attacks [12, 23]. Gnutella has suffered from the query flood DDoS (distributed denial of service) attacks [24]. It is also demonstrated that Overnet can be employed to perform DDoS attack on an ordinary, non P2P host [25]. On the other hand, P2P applications with good security measures gain popularity. For example, BitTorrent includes hashes of the blocks in its torrent file and has successfully fended off the pollution attack. The always encrypted Skype call becomes an important feature that attracts enterprise and residential customers.

3 P2P overlay

After reviewing the widely deployed P2P applications, let us take a look at the key building components of a P2P application. To achieve efficient and robust P2P content delivery, we need to work on two primary modules: P2P overlay construction and P2P scheduling.

All P2P networks run on top of the Internet. We often consider the P2P network as an overlay network, with the link of the overlay being a pair of connected peers. P2P overlay construction is the first important task faced by an architect of a P2P application. Sometimes, the task is called peer matching, because it involves how a new peer finds existing peers and connects to them; and how an existing peer finds replacement peers to substitute those that leave the P2P network.

In this section, we survey the technical approaches and issues in building the P2P overlay network. The target is to have an overlay construction strategy that may efficiently utilize the resource in the network and may effectively deal with the dynamics of the peers and the network conditions. We survey centralized and decentralized approaches in building the P2P overlay network in Section 3.1. We then examine the tiered overlay in Section 3.2. The distributed hash table, an active research area in the distributed system, is covered in Section 3.3. Peer proximity and heterogeneity is covered in Section 3.4.

3.1 Centralized vs. decentralized

The P2P overlay can be built via either a centralized scheme or via a decentralized scheme. In a centralized scheme, a central entity, e.g., a tracker, maintains information of all peers in the P2P network, and is primarily responsible in making decisions on how the peers are matched. An example of centralized overlay building is BitTorrent. The BitTorrent tracker is a central entity that is responsible for building the overlay. It maintains the list of peers that are currently sharing the file. When a new peer joins a sharing session, it finds the address of the BitTorrent tracker through the torrent file, and sends a request (usually through Http and/or Https, but can be UDP as well) to the tracker to discover peers that are sharing the file. The BitTorrent tracker selects among its list of active peers a random set of peers, and returns the peer list, which includes the peer ID, IP address and port used by the peer to the new incoming peer. The default number of peers in the peer list returned by the BitTorrent tracker is 50, though fewer peers may be returned if the size of the file sharing group is small. The new coming peer then attempts to connect to some of the peers in the list, and leaves the addresses of the other in a cache. During the BitTorrent sharing session, the peer maintains the number of connected peers between an upper and lower bound. The default upper bound is 55, and the default lower bound is 30. The BitTorrent peer admits peer connection request until the number of connected peers has reached the upper bound. When the number of the connected peers falls below the lower bound, the peer will try to establish new contacts to peers in the cache; and if the cache becomes empty, the BitTorrent peer contacts the tracker for more peer addresses. It is found that the parameter of the upper and lower bound of the number of connected peers is important to the P2P file sharing performance. If the number of connected peers is too small, the peer may not be able to send and retrieve content efficiently in its local neighborhood. If the number of connected peers is too large, there is a high overhead in exchanging status information among the neighborhood, thus degrades the file sharing performance as well.

An example of decentralized overlay construction is Gnutella. In Gnutella, there is no central entity. The Gnutella peer discovers and connects to other Gnutella peers through a random walk procedure. Each Gnutella node maintains a neighborhood table, which contains the IP address and port of known Gnutella nodes. Similar to BitTorrent, Gnutella has an upper and lower bound of the number of connected peers that it wishes to maintain. When a new Gnutella node comes online, it first finds a bootstrap node. Gnutella uses a number of different mechanisms to bootstrap, including a pre-compiled list of Gnutella nodes that are always online, the use of web caches/UDP caches of known node, or even IRC. The new coming Gnutella node then sends a neighbor discovery message to the bootstrap node with a count of desired neighbors. Upon receiving the neighbor discovery message, the Gnutella node checks if its number of connected peers has reached the upper bound. If not, it connects the new coming node, decreases the count in the neighbor discovery message by one, and if the count is still greater than zero, forwards the neighbor discovery message randomly to one of its neighbors. If the number of the connected peers has reaches the upper bound, it simply forwards the neighbor discovery message randomly to one of its neighbors without decrement the count. As a result, the neighbor discovery message guarantees the new Gnutella node to find the desired number of neighbors, as long as there are enough Gnutella nodes in the P2P network. The Gnutella node also issues the neighbor discovery message to find new neighbors if its number of connected peers falls below the lower bound.

In addition to the pure centralized and pure decentralized overlay building, there are hybrid approaches. For example, in PPLive, the new coming peer first retrieves a list of peers from the tracker. Then, when it connects to other PPLive peers, they further exchange their peer list to discover more peers without involving the tracker.

Let us further take a look at some examples on how other P2P applications build the overlay. CoopNet [29], a P2P broadcast protocol, uses a centralized strategy for P2P overlay building. CoopNet stripes content into multiple streams, and distributes each stream via an independent application-level multicast (ALM) tree. A central server in CoopNet has the full knowledge of the P2P network, and is in charge of assigning insertion points of the ALM tree. NICE [30] and Zigzag [31] use a hierarchy of clusters to form the overlay. Low latency peers are bound into a cluster, in which a lead is selected to represent the cluster. The entire overlay then turns into an ALM tree with the source node being the root during the data delivery. ESM [32] uses a decentralized protocol to self organize end hosts into a mesh overlay. The new coming ESM node first finds a bootstrap node, retrieves a peer list, and then randomly connects to a few more nodes. It then continuously finds better neighbors during the operation, thus evolves the overlay.

Compare with the decentralized overlay building, the main advantage of the centralized overlay building is the simplicity in implementation, and the flexibility to implement additional overlay optimization feature, e.g., optimizing the overlay by peer proximity, heterogeneity and the progress of P2P content delivery. Because the central entity has a global view of the entire P2P network, optimization on the overlay can be easily implemented by using the global knowledge of the P2P network. The main disadvantage of the centralized overlay building is the robustness and scalability. The central entity that is in charge of the overlay building becomes a single point of failure for the P2P network, as its failure leads to the failure of the entire P2P overlay and the subsequent content delivery operation. A single entity also can only support a limited number of peers and file sharing sessions. For example, a BitComet tracker, which is a high performance implementation of a BitTorrent tracker, is reported [28] to capable of support up to 80,000 torrents and up to 800,000 users with the use of a combination of TCP and UDP tracker protocol. If the tracker uses mainly TCP/Http/Https protocol, or involves more complex overlay optimization algorithm, the number of concurrent users that can be supported decreases further. Nevertheless, even with elaborated client-tracker communication and authentication protocol and complicated overlay optimization algorithm, the CPU and bandwidth contributed by the central entity is still a very tiny portion of the CPU and bandwidth resource used by the P2P network. Due to the simplicity of implementation, the centralized overlay building is a popular choice for architects of the P2P applications.

A common method to improve the robustness and scalability of the central tracker is to use multiple central trackers. By establishing multiple trackers for a P2P application, if one tracker fails, the peer may contacts an alternative tracker to continue the service. Each tracker can also be primarily responsible for a subset of peer peers, thus balance the load among them, and allows the tracker cluster to serve a peer crowd beyond the capacity of a single tracker. Multiple central tracker implementation has been widely used in P2P applications, e.g., in multi-tracker torrent of BitTorrent, in eDonkey, etc.. The trackerless BitTorrent and the serverless eDonkey can also be considered as an instance of multiple tracker implementations, except the trackers are automatically identified through node IDs in a global P2P overlay. This is discussed in the following Section.

3.2 Tiered overlay

Tiered overlay is an important method in P2P overlay construction. For example, FastTrack/Skype uses a two-tier P2P network, which can be shown in Fig. 4. The

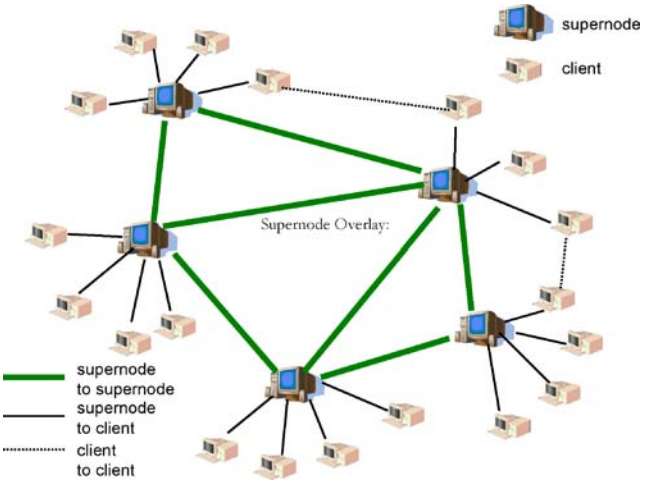

Fig. 4 Two tier P2P overlay used in FastTrack and Skype. The supernode overlay forms the core of the P2P network. Multiple clients connect to the supernode and form the client overlay. During file sharing, two clients may establish a temporarily link for file transfer

supernodes interconnect with the other supernodes and form a supernode overlay which becomes the core of the P2P network. The client node then picks one or a small number of supernodes to associate with and form the client overlay. During the file sharing session in FastTrack or VoIP session in Skype, two clients may temporarily establish a direct connection between themselves to exchange files or conduct VoIP. However, the file search and the user directory query in FastTrack/Skype are served by the supernode overlay only. The method of forming the supernode overlay and the client overlay is proprietary. According to [26, 27], we do know that each supernode connects to around 40–50 other supernodes, and connects to around 50–160 client nodes. The supernode with more bandwidth resource (e.g., on the university campus) also serve more client nodes than the supernode with less bandwidth resource (e.g., on the residential network).

Another common tiered overlay is to let all peer nodes in a particular P2P application form a global overlay, and to let a subset of peer nodes sharing the same content form a small, localized overlay. Such tiered overlay can be shown in Fig. 5. The examples are the trackerless BitTorrent and the serverless eDonkey. In the trackerless BitTorrent, all trackerless BitTorrent nodes sharing all sorts of content form the global P2P network. In the serverless eDonkey, either the Kad network used by the eMule or the Overnet used by eDonkey 2000, all eDonkey nodes form the global P2P network. This large P2P network forms a distributed hash table (DHT), which is the Kademlia DHT implementation of different variants for BitTorrent, eMule and eDonkey 2000, respectively. When a peer joins the P2P network, the peer is assigned with a random ID. Each shared file is also assigned an ID. The list of peers sharing the same content is then stored in those peers with IDs close to the ID of the shared file. In essence, the global DHT network elects some peers with their IDs close to the content ID to be a set of virtual trackers, and involuntarily serve the track functionality. For example, in Azureus BitTorrent, the global P2P network has 1.3 million nodes; and the peer list of a particular sharing session is stored in about 8 peers close to the ID of the session. While in the official BitTorrent, which has 200 thousand nodes, the peer list is stored in about 20 peers close to the ID of the sharing session. A new coming peer wishes to share a certain file may then retrieve the peer list from the DHT. This is equivalent to retrieve the peer list from the virtual tracker cluster. It then connects to peers in the peer list, and form a small, localized P2P overlay for data exchange.

3.3 DHT: Distributed hash table

DHT is an active area in the distributed system research. The first four DHTs, CAN [33], Chord [34], Pastry [35] and Tapestry [36], were introduced in 2001. The common approach of the DHT is to use a large key space, and to let each participating node hold an ID in the key space. Each node is then charged with a set of keys in its neighborhood, as a result, the node can be considered as a slot in the hash table formed by the key space. The resultant distributed system is thus called the DHT (distributed hash table). The DHT design philosophy is to develop a distributed system that can scale to a large number of nodes and can handle constant node arrival and failure. DHT schemes usually implement two basic features: routing and hash table operation. The DHT routing function can be abstracted to: FindNode(ID), which takes an ID as the input, and finds

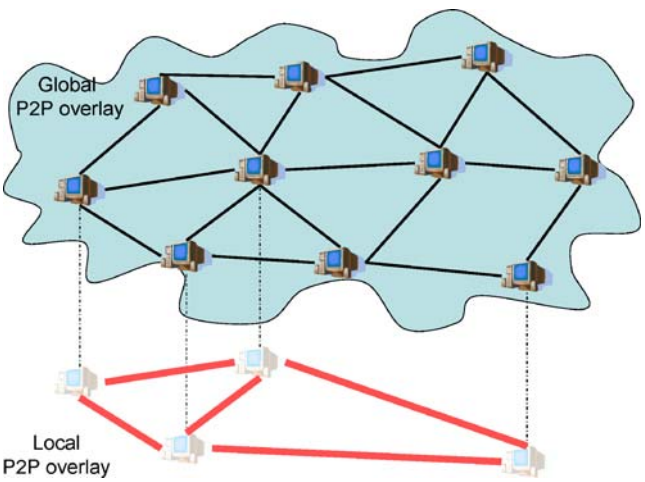

Fig. 5 Global and local P2P overlay. The global overlay is formed by all peer nodes of a particular application, e.g., all peers of the trackerless BitTorrent or the serverless eDonkey. The local overlay is formed by peers that share the same file. The global overlay provides distributed tracker or distributed directory service. The heavy traffic data flow is in the local overlay

one or a set of nodes that are in charge of the space covering the ID. Those nodes are usually the nodes closest to ID with some distance measure. Via routing, DHT may implement two further hash table operations: Store(Key, Value) and Retrieve(Key). In such a case, the DHT can be considered a super reliable and always available hash table, where the Store functionality stores a <Key, Value> pair into the system, whereas the Retrieve functionality retrieves the Value associated with the Key. To implement the Store functionality, the DHT system first uses FindNode(Key) to find a set of nodes covering the Key, and then stores the <Key, Value> pair to each of the nodes. During the Retrieve operation, the DHT uses FindNode(Key) again to find nodes covering the Key, and then retrieves the Value information associated with the Key in each of the node and aggregates the retrieved information. The distributed system formed by DHT guarantees information reliability as even some nodes join, leave or fail in the system, there will be enough nodes that hold the <Key, Value> pair alive and cover the space associated with Key, thus return the Value information associated with Key reliably and consistently. To improve reliability, the DHT system may also migrate to a new node the stored <Key, Value> pair as nodes join and leave the system. Different DHT schemes differ primarily in how the key space is formed, how the routing is performed, how to handle node churn (arrival, departure and failure event), and how to handle proximity and heterogeneity of the node.

A thorough survey, study and comparison of all DHT systems are beyond the scope of this paper. For interested readers, please refer to [37]. In the follows, we explain the Kademlia DHT [38] protocol used in the trackerless BitTorrent and the serverless eDonkey.

In Kademlia, each node holds a 160-bit node ID. The key space in Kademlia is thus $[0, 2^{160}-1]$. The key operation of the Kademlia DHT is FindNode(Key) called upon by a certain node x, which finds k closest nodes to a given Key. To perform the FindNode(Key) operation, the node x first finds k nodes closest to Key in its own routing table, and then picks α closest nodes (default α=3): x_1, x_2 and x_3. The node x then sends FindNode(Key) message to node x_1, x_2 and x_3, and requests each node to return its knowledge of the k closest nodes to Key. If the FindNode (Key) uncovers nodes closer to x_1, x_2 and x_3, the process repeats with the newly discovered node being the target of the inquiry. If no closer nodes are found, the node x expands the search by contacting all k closest nodes, and requests them to return their knowledge of the k closest nodes. If closer nodes are found, they again become the target of the inquiry. The process repeats until no more closer nodes to Key can be found. Kademlia then implements hash table operations Store(Key, Value) and Retrieve (Key) as described above.

In Kademlia, the distance between two nodes is defined by $d(x,y) = x \oplus y$, where x and y are the ID of the node, and $\oplus$ is the XOR operation. Each Kademlia node maintains a routing table. For any node that communicates with the current node, its entry is added to the proper entry of the routing table. And for nodes that repeatedly fail to respond to query, its entry is removed. The Kademlia routing table is organized by sub trees, with each sub tree covers a portion of the ID space sharing the same prefix. For example, sub tree $c_0c_1c_2$... covers all IDs start with $c_0c_1c_2$. Each sub tree is assigned with a routing entry that is implemented as a k-bucket, i.e., a bucket with at most k nodes. At first, the Kademlia node x has only one routing entry, i.e., one k-bucket, that covers the entire key space. When the number of nodes known by node x exceeds k, the original k-bucket is split into two k-buckets of two sub trees 0... and 1.... One of the k-bucket has a different prefix with the node x. This k-bucket is not further split[1]. That is, if a new node y is to be added to the k-bucket and if the bucket has already k nodes, either the new node y or one of the existing node in the k-bucket has to be thrown out. The other k-bucket that has the same prefix with the node x can be further split into two sub trees. Let the ID of the node x be 0011.... The node x may have k-bucket entry for 1..., 01..., 000..., 0010..., 0011...; where the first four entries cover sub trees with prefix differing from the node x by 1 bit, and the last entry is the sub trees with prefix covering the node x. Each Kademlia node has approximately $\log_2(N)$ routing entries, with N being the number of nodes in the P2P network, and each routing entry has at most k nodes.

DHT has been well established in the distributed system. As shown in Section 3.2, DHT has been employed to implement a decentralized tracker for P2P file distribution. However, except in rare cases such as SplitStream [39], it is uncommon to directly use the overlay of DHT as the overlay of content distribution.

3.4 Proximity and heterogeneity

A key issue in overlay building is to consider the proximity and the heterogeneity of the peers. The heterogeneity concerns the difference of the resource available to a peer, e.g., the upload /download bandwidth, the CPU resource, and the NAT/firewall that the peer is connected to. The proximity concerns the distance between the two peers, e.g., the latency, the throughput and the ISP locality. It is beneficial to link neighbor peers in the overlay, and to use peers with more resource as the hubs in the overlay. However, P2P applications may differ in the most crucial proximity and heterogeneity measure. For example, in P2P

[1] Some optimization of Kademlia allows the k-bucket with different prefix to split to a max depth d.

file sharing, a peer may want to find a matching peer with a good throughput and in the same ISP. In P2P VoIP, a client node may want to find a supernode with low latency.

Some examples of heterogeneity and proximity measures are as follows.

a) Latency

Latency is one of the basic proximity measures. The latency between two peers can be easily measured by the ping time, or the message round trip time between the two peers. However, if the P2P network becomes large, it is impractical for each peer to ping every other peer. Virtual coordinate system has been an effective tool to estimate the latency between two arbitrary peers in a large P2P system. The scheme works the following way. First, we find a coordinate system $\mathbf{V^k}$ of k-dimensions, and map each peer to a coordinate. We correspond the latency between the two peers x and y to the distance $d(x,y)$ in the coordinate system. During the construction of the virtual coordinate system, a set of well known hosts are established as reference points. The roundtrip times are measured between the reference points. After that, we map the reference points to a set of coordinates which best preserve the roundtrip time measures obtained. For a new coming node, we just need to measure its roundtrip time to the established reference points, and then find a coordinate for the node which best preserves the measure. Now, the latency between the new node and any node in the system can be measured by calculating the distance in the virtual coordinate systems. Examples of research include the global network positioning (GNP) system [40], the practical internet coordinates (PIC) [41], the Lighthouses [42], the virtual landmark and the internet coordinate system [43]. The quoted approaches above mainly differ in the dimensions of the coordinate system used, the method to handle the distortion of the measure, and the strategy in dealing with the triangle inequality in the coordinate system.

b) ISP Locality

With the booming of P2P content delivery, the internet service provider (ISP) is under tremendous pressure to carry the P2P traffic. It is estimated that 92% of P2P traffic crosses ISP boundaries. As the content servers use P2P technologies to delivery content, their costs are being passed onto the ISPs. To reduce the operation cost of ISP and the traffic on the backbone of the Internet, and to improve the QoS of the end users, it is crucial to develop ISP friendly P2P solution. This involves developing P2P overlay construction that considers the internet topology. In the Internet, the widely exposed entity is the autonomous system (AS), which is a collection of networks and routers under the control of one entity, usually an ISP. Each AS is allocated with a unique AS number (or ASN), which is used in BGP (border gateway protocol) routing. By preferentially matching up peers under the same AS, we may effectively reduce the traffic across the ISP boundaries.

The Internet address to AS mapping information can be obtained by parsing a BGP table dumped from a BGP router. For applications that do not have access to a BGP router, there are dozens of unpruned backbone BGP tables available on the Internet from projects such as Skitter [46] and RouterViews [47]. AS topology mapping project such [48] may be further used to infer the number of AS hops between two Internet addresses.

c) Bandwidth and Throughput Estimation

An important heterogeneous measure of a peer node is its upload bandwidth. Among the upload, download, and link bandwidth measure, the upload bandwidth is also the most important throughput measure. There are several reasons. In terms of the contribution of a peer node to the network, it is the upload bandwidth of the peer node that counts. In a P2P network, we may characterize the network by assigning an upload and a download bandwidth constraint on each peer node, and a link bandwidth constraint between any two nodes or any two groups of nodes. However, the bottleneck is usually the upload bandwidths of the nodes. Since in P2P, a peer node may upload content to multiple destinations, the output of the peer node splits among multiple receivers. As a result, the link bandwidth required between the two peer nodes is only a fraction of the upload bandwidth of the sending node, which usually does not become the bottleneck. In increasingly common networks, the total upload bandwidths of the end-user nodes are much smaller than the total download bandwidths. This is especially true for end-user nodes on the cable modem and ADSL networks, for which the balance is asymmetrically skewed towards larger download bandwidth. Even for user nodes on the campus networks or the corporate networks, the download bandwidth can still be much larger than the available upload bandwidth because the user may cap the upload bandwidth to limit its participation in the P2P activity. As a result, the performance of the P2P application is usually constrained by the upload bandwidths of the peers. The simplest method to measure the upload bandwidth of a peer is to measure the upload TCP throughput of the peer to a well known node on the network with ample download bandwidth. However, such method of measurement is resource intensive, intrusive, and relies on the upload destination to be placed at a server with no bottleneck to the rest of the Internet. The Pathneck algorithm [49] uses a set of low TTL (time-to-live) measurement packets surrounding a sequence of load packets to measure the upload bandwidth and the upload bottleneck of a peer node. It is a good solution to estimate the upload bandwidth without incurring the upload traffic.

Building a proximity and heterogeneity aware overlay is relatively simple for a centralized P2P overlay building

solution. Since the central entity that is responsible for overlay building has global knowledge of the P2P network, it may easily take the proximity and heterogeneity into consideration in building the overlay. The strategy is to record the resource of the new coming peer, and to measure the distances between it and the other peers with a certain proximity measure. The central entity may then match the new coming peer to a nearby peer or a peer with more resource. Example of research in the area can be found in [50, 44].

To build a proximity and heterogeneity aware overlay for decentralized P2P application, the basic idea is to use the proximity neighbor selection (PNS) or the proximity route selection (PRS). Examples of research in the area can be found in [45, 71, 72]. In general, the schemes rely on the DHT to allow multiple peer candidates per each routing entry. Then, in the routing table construction stage, a nearby peer or a peer with more resource is used in the routing table in preference over a far away peer or a peer with less resource.

4 P2P scheduling

The P2P scheduling concerns the method for delivering the data from the source to its destinations under a given overlay. A good P2P scheduling algorithm has three aspects. First, it is efficient in utilizing the bandwidth resource available in the P2P network. Second, it is robust in adapting to the changes in the conditions of the peer and the network. Third, the delivery satisfies certain quality of service requirement of the content.

We examine a number of issues in P2P scheduling. We investigate the tree versus the mesh-based delivery scheme in Section 4.1. We then compare pull, push and hybrid delivery method in Section 4.2. We discuss the peer flow control in Section 4.3. We take a look at the peer and block selection issues in Section 4.4. Finally, the usage of block coding and block mixing in P2P content delivery is discussed in Section 4.5.

4.1 Tree-based vs. mesh-based delivery

We may roughly classify the P2P content delivery methods into two categories: the tree-based delivery and the mesh-based delivery. The tree-based P2P delivery can be traced back to IP multicast [51], where a single block transmitted from the source is replicated by the routers along a distribution tree rooted at the source node, and is thereby delivered to an arbitrary number of receivers. IP multicast utilizes computation and bandwidth resource of the router to replicate and distribute the block, thus, it is not a strict P2P application per se. Though an efficient solution, IP multicast deployment is slow in the real world because of issues such as inter-domain routing protocols, ISP business models (charging for multicast traffic), congestion control along the distribution tree, security and so forth. The reality is that IP multicast is only enabled in small, isolated island on today's Internet. The end user extension of IP multicast is the application-level multicast (ALM). In ALM, each peer in the distribution tree implements all multicast related functionalities including block replication, membership management and content delivery on the overlaid network. Pioneer works in ALM includes ESM [32], Scattercast [52] and Overcast [53]. ESM, Scattercast and Overcast all use a single ALM tree to distribute content, with each non-leaf peer in the tree replicates and forwards blocks upon their arrival. Single ALM tree does not utilize the bandwidth of the leaf peers in the system, and if an intermediate node fails, the peers under the failing sub tree may not receive content before the overlay is rebuilt. CoopNet [29] and SplitStream [39] split the content into multiple stripes and distribute the stripes across separate ALM trees with disjoint interior nodes. Any peer computer can be an interior node in one of the multicast trees, and contributes to forwarding the content. CoopNet further utilizes multiple description coding (MDC) and forward error correction (FEC) to protect from block loss and node failure. MutualCast [54] and FastReplica [55] push the tree-based delivery to an extreme. In a P2P network of one source and n peer nodes, FastReplica uses n trees in delivering the content, and MutualCast uses a total of $n+1$ trees (n trees with the interim node being one of the receiving peers, and 1 tree from source to all peers) to delivering the content. MutualCast further uses the TCP buffer to adapt the bandwidth of each delivery tree based on the available upload bandwidth of each peer node, thus efficiently and adaptively delivers the content in synchronous fashion to all destinations. The primary feature of the tree-based P2P content delivery is that each block traverses a deterministic route from the source to its destinations based on the overlay. Once the overlay is determined, the traverse paths of the blocks are fixed during the delivery. If the peer and network condition changes, the tree-based overlay generally needs to re-adjust the overlay to achieve the optimal efficiency. The exception is MutualCast, which re-adjusts the rate assigned to each delivery tree in response to the change.

In contrast, in the mesh-based P2P delivery, the delivery path of the block is based upon not only the overlay, but also the feedback from the neighbor peers. It is customary in the mesh-based delivery to create an over redundant overlay. During the content distribution, each peer receives status information from its directly connected neighbors, and makes a distributed decision on how the content should flow in the overlay. BitTorrent [14] and PPLive [17] are examples of the mesh-based delivery in action.

Compare the tree-based with the mesh-based P2P delivery, the mesh-based delivery can more robustly cope with the change in the peer and network conditions, and more efficiently utilize the peer resource. The tree-based delivery relies on the adaptation of the overlay to adjust the block distribution path, while the mesh-based delivery relies on the exchanged status information to redirect the flow of the blocks. It is apparent that the latter is easier to implement and more timely reflects the network and peer condition. To achieve efficient and reliable content delivery, the tree-based delivery scheme needs to use a dense distribution forest such as the one used in MutualCast [54], which is difficult to scale to a large number of nodes. The primary advantage of the tree-based delivery is in the reduced latency in delivering the content. In tree-based delivery, each intermediate node may immediately forward a received block to its downstream peers. In comparison, the mesh-based delivery has to delay the forward of the received block until status information has been gathered to make sure no duplicate delivery is performed. As a result, the tree-based delivery is more suited for delay bounded broadcast P2P applications.

4.2 Pull vs. push vs. hybrid delivery

Depend on whether the sender and/or the receiver take the initiative in moving the blocks, three modes may be used in the P2P delivery. In the push mode (also called the sender-driven delivery mode), the sender takes the initiative, and pushes the received block to a selected peer. In the pull mode (also called the receiver-driven delivery mode), the receiver takes the initiative, and pulls the block it wants from its neighbor. In the hybrid mode, both the receiver and the sender may take the initiative, and negotiate the block delivery.

The tree-based P2P delivery usually adopts a push based delivery mode. The peer simply pushes the arriving blocks towards its downstream peers in the tree. The mesh-based P2P delivery is more flexible, and may use either the pull, push or combined delivery mode. For example, BitTorrent adopts a pull based delivery method. A BitTorrent node exchanges with its neighbor peers the *have* information, i.e., what blocks it holds, and make a decision on which block to retrieve based on the available blocks in its neighborhood. DISCOVR [56], a P2P delivery platform combines file sharing and media streaming, uses a push mesh-based delivery method. The peer gathers information on what the neighbor peers have, and propose a set of blocks that it wishes to send to the neighbor peers. Upon receiving the list, the neighbor peer chooses to reject certain proposed blocks if they are already proposed or are on the way from the other senders, and confirms the rest. The actual block delivery is then initiated by the sender after receiving the confirmed list. Avalanche [57], a P2P file sharing tool using the network coding to mix blocks during the delivery, is also a push mesh-based delivery protocol. In Avalanche, the sender first proposes to the receiver the code vector of the transmitted block. The receiver checks whether the code vector contains new information, and accepts or rejects the code vector. The sender then sends the network coded block after the confirmation. GridMedia [58], an early P2P streaming platform, adopts a hybrid push-pull mesh-based delivery. The media blocks are classified into pushing blocks and pulling blocks. The node uses the pull mode when it first joins the network, and then uses the push mode to redelivery blocks to its neighbors for reduced delay.

We note that the push mode used in the tree-based delivery is rather different from the push mode used in the mesh-based delivery. The former doesn't need to consult the receivers for the validity of the block, and may thus achieve low delay block delivery. We also observe that in the mesh-based delivery, the push and the hybrid modes are more efficient in utilizing the upload bandwidths of the P2P network. Because in either mode, the sender may take the initiative, it can easily make sure that the sender's available upload bandwidth is fully utilized. The pull based mode is more beneficial to maintain the quality of service (QoS) for the receiver. Because the receiver can select blocks to receive, it has a better control of the delivery process, and may more easily ensure the on time delivery of a particular block.

4.3 Flow control

The flow control is an essential part of a P2P delivery algorithm. For the sender, the flow control ensures that the upload pipeline is fully utilized. For the receiver, it ensures that a constant stream of blocks is arriving. P2P applications employ different flow control mechanisms. For example, certain BitTorrent implementation suggests [59] keeping a few unfulfilled requests on each connection in order to avoid a full round trip interval of empty delivery pipeline from the finishing of the download of one block to the beginning of the download of another block. PeerStreaming [60], an on-demand P2P streaming platform with pull mesh-based delivery, use a flow control strategy that keeps the pipeline from each sender filled to the same round trip interval between the request and the reply. BulletPrime [61] uses a XCP-like [62] congestion control scheme to maintain one waiting block in the sending TCP. Examining these schemes, we notice two common traits of a good flow control scheme in the P2P delivery. First, it keeps the upload pipeline of the sending peer busy, so that no upload bandwidth of the sending peer is wasted. Second, it keeps the queuing delay between the sender and the receiver low, so that the peer can quickly respond to the condition change in the peer and the networks.

4.4 Peer and block selection

In either the push or the pull mode, the peer needs to decide upon a delivery strategy, i.e., which block to push/pull, and from/to which peer that the block is pushed and pulled. We call this the block selection and the peer selection.

The peer selection chooses the peer that the block is to be pulled from or pushed to. For the pull mode, the receiver usually selects the sending peer based on the peer flow control. For example, PeerStreaming [60] selects to pull from the peer that offers the lowest roundtrip delay between the time of the request to the time of the reply. For the push mode, the sender may select the receiving peer based on a fairness measure or a performance measure. For example, the BitTorrent uses a fairness measure, which is the Tit-for-Tat strategy. The BitTorrent peer preferentially uploads to top m (default m=5) neighbors that provide the peer with the best download rate. This prevents free riding, and encourages the peers to contribute. However, the peer nodes with inherent low resource (low upload bandwidth) may suffer from low delivery throughput. An alternative peer selection strategy is based on the performance measure. For example, DISCOVR [56] allocates more sending bandwidth to the peer that is running low in media playing buffer. Such peer selection strategy allows high resource peers to subsidize low resource peers, and improves the QoS level in the overall system.

The block selection chooses one block among a list of blocks that are pushable or pullable from a particular neighbor. Some possible block selection schemes [61] are:

a) Sequential: Choose the first block that is pushable or pullable.
b) Rarest: Choose the block that is the rarest in the local neighborhood of the peers. There is no method for tie breaking if there are multiple blocks that are equally rare. BitTorrent uses the rarest selection strategy to pull blocks.
c) Random: Select a random pushable or pullable block.
d) Random rarest: Choose the block that is the rarest in the local neighborhood of the peers. If multiple blocks are of equal rarity, a random block is selected.

Results from BitTorrent and BulletPrime [61] show that the random rarest selection strategy performs the best in terms of efficiency, the random selection or the rarest selection trail close behind, while the sequential block selection strategy performs very poor.

4.5 Block coding and network coding

In the above, we assume that the content to be delivered is simply split into a set of blocks, each of which is then sent in its original form into the overlay. In 1972, Edmonds [64] established a fundamental graph theorem that if all nodes other than the source are destinations, the delivery capacity can be achieved by routing. Thus, for the P2P content delivery that does not use IP multicast and with all nodes being either the source or the receiver, routing can achieve the network capacity. Theoretically, there is no gain for the use of advanced coding technology.

Good P2P content delivery algorithms such as BitTorrent do demonstrate remarkable efficiency during the bulk of the content delivery period. Nevertheless, it has been shown [57] that coding can be still used to improve the content delivery throughput at the beginning of the P2P session and at the end to resolve the "last coupon collector" problem if the source node leaves the content delivery session in the interim.

To use erasure coding in the P2P content delivery, we usually group a set of k blocks into a generation, and apply erasure coding on each generation separately. Note that it is possible to use just one generation to cover the entire file/media to be delivered. However, since this may result in many blocks and a long content file, it may greatly increase the CPU and memory complexity of erasure encoding/decoding. Erasure coding can be applied in one of two ways. It can be applied just on the source node. That is, using either a digital fountain [66], a Reed-Solomon [67] or a random linear erasure code, we expand the original k blocks into n (n is much larger than k) coded blocks at the source node. The source node then ensures a distinct coded block is delivered every time to its directly connected peer cluster. Each directly connected peer further forwards the coded blocks to its down streaming peers and so on, without peers mixing or further coding the blocks. For each generation of coded blocks, only k blocks[2] need to be received by each peer to recover all the original k blocks. Since in this scheme, the erasure coding is only used at the source node to expand the number of distinct blocks of each generation from k to n, we call this the block coding approach. BulletPrime [61] has used the block coding for P2P file sharing.

If the intermediate peers further mix the coded blocks via a random linear code, network coding is used in the P2P delivery. The effectiveness of network coding in improving the network delivery capacity is first established in [65], and then ported by Avalanche [57] for P2P file sharing. With network coding, each time a block is to be delivered from the source node to its directly connected peers, the source node generates a random vector $\mathbf{g}$, encodes the original k blocks with the vector, and sends the encoded block $\mathbf{b}$ with the coding vector $\mathbf{g}$ to the peer. During content delivery, the intermediate node may mix and redeliver content. Let an intermediate node receive m blocks $\mathbf{b_1}$,

[2] Since Reed-Solomon is a maximum distance separable (MDS) erasure code, only exact k distinctive coded blocks are needed. If fountain code or random linear code is used, we may need $(1+\varepsilon)k$ blocks, with ε being a very small number for random linear code, and being around 0.03–0.25 for fountain code.

$\mathbf{b_2}$,...,$\mathbf{b_m}$ with coding vectors $\mathbf{g_1}$, $\mathbf{g_2}$,...,$\mathbf{g_m}$. It may mix and regenerate a new block **b'** with coding vector **g'** as:

$$\mathbf{b}' = \sum_{i=1}^{m} \alpha_i \mathbf{b_i},$$

$$\mathbf{g}' = \sum_{i=1}^{m} \alpha_i \mathbf{g_i},$$

where α_i, i=1,...,m are random non-zero values in the Galois Field. With the network coding, each peer needs only to receive a little more than k blocks to decode the original set of k blocks[3].

Both the block coding and the network coding can be used to improve the content delivery throughput at the beginning of the content delivery session. Because distinct blocks are sending from the source node, all peers directly connected to the source can forward useful information to its neighbors, thus eliminate a potential bottleneck at the source node. They can also resolve the "coupon collector" problem during the end of the delivery session. Either approach greatly increases the number of distinct blocks in the P2P system, and makes the job of finding the last distinct block easier. The network coding does create more block diversity by mixing and generating new blocks.

One cautious note is that the network coding is more susceptible to pollution attacks [12]. Because the network coding generates new version of the block at the intermediate node, traditional counter pollution strategy of signing the block at the source node or pre-delivering the hash of the block in a secured channel (e.g., in the torrent file of BitTorrent) is not effective any more. We need to either use a homomorphic hash [68] to verify the hash of the mixed block, or use a homomorphic signature [70] to sign the mixed blocks. Either approach greatly increases the computation complexity and implementation complexity, as it requires a much larger Galois Field for security reasons. An alternative is to use a cooperative protection method [69]. The approach [69] doesn't increase the computation complexity that much. However, it requires an authentication server to always be online to generate an authenticated random mask for the new coming peer. This can not solve the "coupon collector" problem that leads to the system failure if the authentication server leaves the system.

[3] Avalanche 0 uses a push based confirmation process to ensure exact k blocks are needed to decode the original. In Avalanche, the sender first proposes to the receiver the code vector of the transmitted block. The receiver checks whether the code vector is linearly independent to the code vectors of the already received blocks, and accepts or rejects the code vector. This way, no bandwidth is wasted. Nevertheless, it may increase the latency in the delivery.

5 Summary and future works

In this paper, we survey the current state of the art of P2P content delivery. We establish that a good P2P system should reliably and efficiently deliver the content and satisfy certain quality of service (QoS) constraint. We examine P2P file sharing systems Gnutella, FastTrack, eDonkey and BitTorrent, and P2P streaming systems PPLive, as well as many research works on P2P file sharing, P2P streaming, and P2P broadcast. Using these systems as examples, we investigate good practices for P2P overlay construction and P2P scheduling, and discuss methods that can support efficient and reliable P2P content delivery.

Currently, the research and the development in P2P file sharing solution are mature. Existing P2P file sharing applications, such as BitTorrent [14], have demonstrated that they can efficiently and reliably transfer the file in P2P fashion to a huge pool of users in the real world.

Though there are a number of deployed P2P streaming solutions such as PPLive [17], and there are many P2P streaming prototypes, the problem of efficiently and reliably streaming content in P2P is still not solved. For example, it is shown [22] that PPLive cannot sustain the streaming to all peers without playback stalling. There are still many research works to be done to use the network resource during streaming more efficiently, to sustain the streaming rate more robustly in case of peer and network anomaly, and to make sure that the playback does not pause.

The research in the P2P broadcast concentrates around tree-based P2P delivery schemes derived from the application layer multicast (ALM). The best available P2P broadcast systems are not as efficient, as robust, and as scalable compared to solutions in the P2P file sharing and P2P streaming space. It will be interesting to see more works in the P2P broadcast and/or P2P conferencing space.

Another interesting research area is the hybrid mode P2P delivery. Currently, most P2P applications serve content in a single mode, either in file sharing, in streaming, or in broadcast. It is rare to see works such as [56], which supports both file sharing and streaming. A hybrid P2P content delivery application may aggregate resource in the P2P network and improve the QoS for users in the streaming and the broadcast mode. Let us imagine a P2P platform that supports file sharing, audio/video conferencing (VoIP), and music/video streaming simultaneously. The user of such P2P platform tends to always be online, as it can be used for so many tasks, and the file sharing task alone can take a long time. This P2P system is capable of diverting the resource of the P2P network to ensure that those peers with stringent QoS demand are met first. It may improve the QoS experience of the users beyond what is capable of by a single mode P2P content delivery system.

P2P applications are being adopted at record speed. It is estimated that 65–70% of the Internet backbone traffic can be attributed P2P. If the majority of the P2P applications are agnostic to the Internet architecture, they will quickly overwhelm the Internet. It is crucial to build P2P applications that are aware of the underlying Internet architecture, and are friendly to the ISPs. Some works along the directions are [74–76].

The most popular commercial solution of large scale content delivery is the CDN (content delivery networks). CDN deploy servers in multiple backbones and ISPs, and often in multiple POPs (point of presences) within each ISP. By providing a shared distribution infrastructure, CDNs provide reliable delivery and cost-effective scaling. P2P serves as a natural complement to CDN, with the main difference being that P2P employ user peers not managed by the CDN directly. The concept of using P2P to extend the CDN network has been raised by many researchers, e.g.,[77, 78]. Akamai, with its acquisition of Red Swoosh, expects to combine P2P file management and distribution software with the scalable backend control system and global network of edge servers of Akamai. VeiSign [79], CacheLogic, Grid Networks and Joost all announce their own P2P CDN service as well. However, none of the existing works analyze the key building blocks of P2P-CDN, as well as quantifying the performance difference, in term of latency and throughput, among the P2P, CDN and hybrid P2P-CDN solutions.

References

1. A&M Records, Inc vs. Napster Inc, United States District Court, Northern District of California, Mar. 2001
2. "Testimony of Mr. Shawn Fanning", United States Senate Committee on the Judiciary, hearings on the Utah's digital economy and the future: peer-to-peer and other emerging technologies. Oct. 9, 2000
3. Report: Napster users lose that sharing feeling," in CNN news, URL: http://archives.cnn.com/2001/TECH/Internet/06/28/napster.usage/
4. Ferguson D (2006) Trends and statistics in peer-to-peer. Workshop on technical and legal aspects of peer-to-peer television. Amsterdam, Netherlands, Mar. 17
5. Winer D. Clary shirky on P2P. URL: http://davenet.scripting.com/2000/11/15/clayShirkyOnP2p
6. Schollmeier R (2001) A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Proc. of 1st Intern. Conf. on Peer-to-Peer Computing, Linkoping, Sweden, pp. 101–102, Aug
7. Sundsted T. The practice of peer-to-peer computing: Introduction and history." URL: http://www-106.ibm.com/developerworks/java/library/j-p2p
8. Steve Saporta, "Multiple bit rate streams", Jun. 1, 2002, URL: http://www.midicorp.com/articles.php?section=articles&id=24
9. Ripeanu M (2002) Peer-to-peer architecture case study: Gnutella network. Internet2 workshop: collaborative computing in higher education: peer-to-peer and beyond. Tempe, Arizona, January 30–31
10. Ritter J. Why Gnutella can't scale. No, really. URL: http://www.darkridge.com/~jpr5/doc/gnutella.html
11. Llie D (2006) Gnutella network traffic measurements and Characteristics. Licentiate Dissertation Series No. 2006:05, ISBN: 91-7295-084-6, April
12. Liang J, Kumar R, Xi Y, Ross KW (2005) Pollution in P2P file sharing systems. Proc. of INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, 13–17 March, Miami, FL
13. Hoßfeld T, Leibnitz K, Pries R, Tutschku K, Tran-Gia P, Pawlikowski K (2004) Information diffusion in eDonkey file sharing networks. University of Würzburg, Research Report No. 341, Sept
14. Comparison of BitTorrent software. From Wikipedia, URL: http://en.wikipedia.org/wiki/Comparison_of_BitTorrent_software
15. RawFlow Ltd., URL: http://www.rawflow.com/
16. Abacast Online, URL: http://www.abacast.com/
17. PPLive.com, URL: http://www.pplive.com/
18. PPStream, URL: http://www.ppstream.com/
19. UUSee, URL: http://www.uusee.com/
20. Roxbeam media network, URL: http://www.roxbeam.com/
21. Mysee, URL: http://www.mysee.com/
22. Hei X, Liang C, Liang J, Liu Y, Ross KW (2006) Insights into PPLive: a measurement study of a large-scale P2P IPTV system. In: Workshop on Internet Protocol TV (IPTV) services over World Wide Web in conjunction with WWW2006, Edinburgh, Scotland, May
23. Liang J, Naoumov N, Ross KW (2006) The index poisoning attack in P2P file-sharing systems. Proc. of INFOCOM 2006. 25th Annual Joint Conference of the IEEE Computer and Communications Societies, Barcelona, Spain, Apr
24. Daswani N, Garcia-Molina H (2002) Query-flood DoS attacks in Gnutella. 9th ACM Computer and Communications Security Conference (CCS), Washington, Dc, Nov
25. Naoumov N, Ross KW (2006) Exploiting P2P systems for DDoS attacks. International workshop on peer-to-peer information management (keynote address). Hong Kong, May
26. Liang J, Kumar R, Ross KW (2005) The Kazaa overlay: a measurement study. Computer networks: special issue on overlays
27. Guha S, Daswani N, Jain R (2006) An experimental study of the Skype peer-to-peer VoIP system. In: Proceedings of the IPTPS'06. Santa Barbara, CA, Feb
28. BitComet tracker overview, URL: http://www.bitcomet.com/tracker/index.htm
29. Padmanabhan VN, Wang HJ, Chou PA, Sripanidkulchai K (2002) Distributing streaming media content using cooperative networking. ACM NOSSDAV, Miami Beach, FL, USA, May
30. Banerjee S, Bhattacharjee B, Kommareddy C (2002) Scalable application layer multicast. In: Proceedings of ACM SIGCOMM 2002. Pittsburgh, PA, August
31. Tran D, Hua K, Do T (2003) Zigzag: an efficient peer-to-peer scheme for media streaming. Proc. of INFOCOM 2003, the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Francisco, CA, Apr
32. Chu Y, Rao SG, Seshan S, Zhang H (2002) A case for end system multicast. IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast 20(8)
33. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S (2001) A scalable content-addressable network. In: Proceedings of ACM SIGCOMM 2001. San Diego, CA, Aug
34. Stoica I, Morris R, Karger D, Frans Kaashoek M, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: Proceedings of ACM SIGCOMM 2001. San Diego, CA, pp 149–160, August
35. Rowstron A, Druschel P (2001) Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). Heidelberg, Germany, pp 329–350, November

36. Zhao BY, Huang L, Stribling J, Rhea SC, Joseph AD, Kubiatowcz JD (2004) Tapestry: a resilient global-scale overlay for service deployment. IEEE J Sel Areas Commun 22(1), Jan
37. Ylä-Jääski A, Kasinskaja N (eds) (2005) Peer-to-peer technologies, networks and systems, Helsinki University of Technology, T-110.551 Seminar on Internetworking in the spring
38. Maymounkov P, Mazières D (2002) Kademlia: a peer-to-peer information system based on the XOR metric. In: 1st international workshop on peer-to-peer systems (IPTPS 2002). Cambridge, MA, Mar
39. Castro M, Druschel P, Kermarrec A-M, Nandi A, Rowstron A, Singh A (2003) SplitStream: high-bandwidth multicast in a cooperative environment. SOSP'03, Lake Bolton, New York, October
40. Ng TSE, Zhang H (2002) Predicting Internet network distance with coordinates-based approaches. Proc. of INFOCOM 2002, the 21st Annual Joint Conference of the IEEE Computer and Communications Societies, New York, NY, June
41. Costa M, Castro M, Rowstron A, Key P (2004) Practical internet coordinates for distance estimation. Proc. ICDCS 2004, the 24th International Conf. on Distributed Computing Systems, Tokyo, Japan, Mar
42. Pias M, Crowcroft J, Wilbur S, Bhatti S, Harris T (2003) Lighthouses for scalable distributed location. In 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03). Berkeley, CA, Feb
43. Tang L, Crovella M (2003) Virtual landmarks for the internet. Proc. of the ACM/SIGCOMM Internet Measurement Conference 2003. Miami, FL, pp 143–152, Oct
44. Bindal R, Pei C, Chan W, Medved J, Suwala G, Bates T, Zhang A (2006) Improving traffic locality in BitTorrent via biased neighbor selection. Proc. ICDCS 2006, the 26th IEEE International Conference on the Distributed Computing Systems, Lisboa, Portugal, July
45. Li J, Sollins K (2004) Exploiting autonomous system information in structured peer-to-peer networks. Proc. ICCCN2004, the 13th IEEE International Conference on Computer Communications and Networks, Chicago, IL, October
46. Skitter, URL: http://www.caida.org/tools/measurement/skitter/
47. RouterViews, URL: http://www.routeviews.org/
48. Dimitropoulos X, Krioukov D, Riley G, Claffy KC (2006) Revealing the autonomous system taxonomy: the machine learning approach. Proc. Passive and active measurements workshop (PAM). Adelaide, Austria, Mar
49. Hu N, Li LE, Mao ZM, Steenkiste P, Wang J (2004) Locating Internet bottlenecks: algorithms, measurements, and implications. Proc. SIGCOMM 2004. Portland, OR, Sept
50. Setton E, Noh J, Girod B (2006) Low latency video streaming over peer-to-peer networks. Proc. ICME 2006, IEEE Int. Conf. Multimedia and Expo. Toronto, Canada, July
51. Internet protocol multicast, ch 43 of Internetworking Technology Handbook, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/index.htm
52. Chawathe Y (2000) Scattercast: an architecture for internet broadcast distribution as an infrastructure service. PhD thesis, University of California, Berkeley, August
53. Jannotti J, Gifford DK, Johnson KL, Kaashoek MF, O'Toole Jr. JW (2000) Overcast: reliable multicasting with an overlay network. In Proc. of the Fourth Symposium on Operating System Design and Implementation (OSDI), October
54. Li J, Chou PA, Zhang C (2005) Mutualcast: an efficient mechanism for content distribution in a P2P network. Proc. Acm Sigcomm Asia Workshop. Beijing, China, Apr. 10–12
55. Cherkasova L, Lee J (2003) FastReplica: efficient large file distribution within content delivery networks. In: Proc. of the 4-th USENIX Symposium on Internet Technologies and Systems. Seattle, Washington, March 26–28
56. Huang C, Li J (2006) DISCOVR: distributed collaborative video recorder. 2006 International Conference on Multimedia & Expo (ICME'2006), Toronto, Canada, Jul. 9–12
57. Gkantsidis C, Rodriguez P (2005) Network Coding for Large Scale Content Distribution. Proc. of INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Miami, FL, 13–17 March
58. Zhang M, Luo JG, Zhao L, Yang SQ (2005) A peer-to-peer network for live media streaming using a push-pull approach. Proc. ACM Multimedia 2005, Singapore, pp 287–290, Sept
59. BitTorrent Specification, URL: http://wiki.theory.org/BitTorrentSpecification
60. Li J, Cui Y (2007) PeerStreaming: design and implementation of an on-demand distributed streaming system. To be published in ACM Multimedia Systems Journal
61. Kostic D, Braud R, Killian C, Vandekieft E, Anderson JW, Snoeren AC, Vahdat A (2005) Maintaining high-bandwidth under dynamic network conditions. Proceedings of 2005 USENIX Annual Technical Conference (USENIX 2005). April
62. Katabi D, Handley M, Rohrs C (2002) Internet congestion control for high bandwidth-delay product networks. In: Proceedings of ACM SIGCOMM, August
63. Microsoft Research ConfXP, URL: http://www.learningwebservices.com/
64. Edmonds J (1973) Edge-disjoint branchings. In: Rustin R (ed) Combinatorial Algorithms. Academic Press, NY, pp 91–96
65. Chou PA, WuY, Jain K (2003) Practical network coding. Proc. 51st Allerton Conf. Communication, Control and Computing, Oct
66. Byers JW, Luby M, Mitzenmacher M, Rege A (1998) A digital fountain approach to reliable distribution of bulk data. Proc. ACM SIGCOMM. Vancouver, BC, Sept
67. Li J (2005) The efficient implementation of Reed-Solomon high rate erasure resilient codes. Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (2005). Philadelphia, PA, Mar. 19–23
68. Krohn M, Freedman MJ, Mazières D (2004) On-the-fly verification of rateless erasure codes for efficient content distribution Proc. IEEE Symposium on Security and Privacy, Oakland, CA, May
69. Gkantsidis C, Rodriguez P (2006) Cooperative security for network coding file distribution. IEEE/INFOCOM'06. Barcelona, April
70. Charles D, Jain K, Lauter K (2006) Signatures for network coding. Conf. on Information Sciences and Systems, Mar
71. Gummadi K, Gribble S, Ratnasamy S, Shenker S, Stoica I (2003) The impact of DHT routing geometry on resilience and proximity. Proc. Proc. ACM SIGCOMM'03. Karlsruhe, Germany
72. Castro M, Costa M, Rowstron A. Performance and dependability of structured peer-to-peer overlays. Technical report MSR-TR2003-94
73. Huang G (2007) PPLive – a practical P2P live system with huge amount of users. keynote speech at P2PTV workshop, Kyoto, Japan
74. Li J (2007) Locality aware peer assisted delivery: the way to scale Internet video to the world. 16th Packet Video workshop (PV 2007). Lausanne, Switzerland, Nov
75. Karagiannis T, Rodriguez P, Papagiannaki K (2005) Should Internet service providers fear peer-assisted content distribution? In: Proc. Internet Measurement Conference 2005. Berkeley, CA, Oct
76. Huang C, Li J, Ross K (2007) Can internet video-on-demand be profitable. Proc. ACM Sigcomm'07. Kyoto, Japan, Aug
77. Pakkala D, Latvakoski J (2005) Towards a peer-to-peer extended content delivery network. 14th IST Mobile & Wireless Communications Summit, Dresden, June
78. Xu D, Chai HK, Rosenberg C, Kulkami S (2003) Analysis of a hybrid architecture for cost-effective streaming media distribution. SPIE/ACM Conf. on Multimedia Computing and Networking (MMCN'03). San Jose, CA, Jan
79. The VeriSign Intelligent CDN, VeriSign, 2007

Dr. Jin Li is currently a principal researcher managing the communication system subgroup at Microsoft Research (Redmond, WA). He received the Ph.D. with distinction from Tsinghua University (Beijing, China) in 1994. Prior to joining Microsoft in 1999, he has worked at the University of Southern California (Los Angeles, CA) and the Sharp Laboratories of America (Camas, WA). From 2000, Dr. Li has also served as an adjunct professor at the Electrical Engineering Department, Tsinghua University (Beijing, China). His research interests cover audio/image/video/graphic compression, audio/video streaming, realtime audio/video conferencing, peer-to-peer content delivery, distributed storage, etc. Dr. Li has published 80+ referred conference and journal papers. He is currently an Area Editor for the Journal of Visual Communication and Image Representation and an Associate Editor for the Peer-to-Peer Networking and Applications. He has served as an Associate Editor for IEEE Trans. on Multimedia, and on numerous TPC committees for major conferences. He was the recipient of the 1998 Young Investigator Award from SPIE Visual Communication and Image Processing.