Lecture 18: Peer-to-Peer applications

Anirudh Sivaraman

2018/11/19

In this lecture, we'll talk about peer-to-peer (P2P) applications. These are applications in which all end hosts are treated equally: there is no dichotomy between designated servers that are responsible for hosting the content and designated clients that sporadically access this content. Instead, every end host functions as both a client and a server in the sense that it is responsible for both hosting the content and may also choose to access it.

The rationale behind these P2P applications is that you do not have to rely on a few dedicated servers, which may be costly to own and operate. Instead, the costs of running an application could, in principle, be shared equally across all end hosts. As the application grows to include more hosts, the application could scale naturally as these hosts could be put to use in running the application itself. In contrast, in a traditional client-server setup, the number of clients far exceeds the number of servers, and increasing the clients results in an increased load on the small number of servers—as opposed to spreading out the increased load over the entire system. Another motivation for the P2P architecture was the democratization and decentralization of content creation and publishing. A group of users could get together and use their desktops or laptops to voluntarily and cooperatively host content for all these users for free without relying on a costly hosting service.

Examples of P2P applications are P2P versions of hash tables [7, 5, 6, 10], file systems [9, 4], citation databases [8], file sharing services [2], and music search services [1]. More generally, there was an active area of research in the early to mid 2000s in taking any application that was traditionally implemented using the client-server model and figuring out how to implement it in the P2P model without dedicated clients and servers.

Many of these research ideas have not made it into practice in a big way—at least not for a long period of time. Bitcoin is a notable exception. However, Bitcoin seems to be moving towards a centralized model for mining. So it's hard to say whether it is truly distributed or decentralized. One reason is that while volunteer-run applications are an enticing idea in theory, the incentives typically aren't there in practice. This is in contrast to the client-server model where there is an obvious incentive: profit motive. That said, some of these ideas have made it into distributed applications that run *within* the server farms of large web-scale companies partly because these server farms start looking a bit like a tightly integrated low-latency network for running P2P applications. For instance, the Amazon Dynamo system [3] employs a distributed/P2P hash table, which was first considered by Chord [7], CAN [5], Pastry [6], and Tapestry [10].

In this lecture, we'll start with the history of P2P applications and then talk about BitTorrent, one of the most popular P2P applications of the 2000s. This decade, however, BitTorrent has faded in popularity particularly in the developed world due to the rise of other client-server alternatives (it still sees some use in the developing world); we'll explain why that happened.

1 The history of P2P

P2P applications are rooted in the online forums of the 1990s: Usenet and Internet Relay Chat. These were low-throughput applications that were mostly *conversational*: they allowed one user to communicate with one or more other users, but didn't strive to emulate traditional client-server applications such as file sharing or audio/video streaming.

The first P2P application was likely Napster [1], which was created in 1999 as a means for users to cooperatively share and find music on another user's machine. Napster came with a search interface that allowed Napster users to search for music by artist, title, and so on.

Napster soon ran into problems that were an early sign of problems that would plague most P2P applications: the sharing of illegal content. The metal band, Metallica, filed a lawsuit in 2000 against Napster because they discovered that a demo of their song was circulating on the Internet even before it was officially released! As a counterpoint, several musicians argued that Napster had promotional power for unheralded musicians who might lack the marketing resources of an established music band [1].

College campuses also started blocking Napster because it generated large volumes of network traffic [1]. Client-server applications solved the problem of increased network traffic through techniques such as caching—both at the clients themselves and at replicated servers close to the clients. Many early P2P applications implemented neither and hence placed an undue burden on ISPs and campus networks to carry their traffic.

2 BitTorrent

The illegal content problem caused Napster to eventually cease operation as a P2P music sharing and search application.[1] However, Napster was the first of many more P2P applications to come, many of which still suffered from the illegal content problem. One of these was BitTorrent [2], a file sharing program that expanded beyond music to include (sometimes illegal) video games, malware, music, video episodes of various TV shows, videos of various movies, software, and so on.

BitTorrent's interface was simpler than Napster. It provided no way to search for a file within the BitTorrent system. Instead, a user interested in a particular file found out whether and where it was available in the BitTorrent system through a separate mechanism that was outside the BitTorrent system. This mechanism was called a torrent file. The torrent file contained the name of the file that it was responsible for and the URL of a tracker server. If a BitTorrent user (called a peer) contacted the tracker server, the tracker server would tell it the list of other peers who were currently storing different pieces of the file (and which file pieces each of these peers was storing). The user interested in downloading the file would then download different file pieces by contacting different peers. The user would then assemble the file using these pieces. The torrent file itself could be distributed over email or downloaded from some other web server.

Let's see how BitTorrent works in a bit more detail now. Let's assume a BitTorrent user starts out with no pieces of the file she is interested in. What is the right strategy to fetch these pieces? We could fetch the first piece (first by location of the piece in the file) similar to TCP, which provides an in-order reliable bytestream and is the most common way to download files in the client-server model.

However, in a P2P system such as BitTorrent, there is no guarantee that the peers stick around indefinitely; they can come and go as they please. Thus, it is important to first get the file piece that is stored at the least number of peers. This way even if the small number of peers storing that piece leave, someone else has the piece and can continue sharing it for future users. This strategy also makes sense even if the user is only interested in their own self interest: if as a user you want to eventually get all pieces of a file, you are better off first getting the pieces that are most likely to disappear. Hence, BitTorrent follows a rarest first strategy (i.e., the piece stored at the least number of peers) to download pieces among all the pieces that are currently undownloaded.

The flip side of this rarest first strategy is that the file isn't usable until all the pieces have been downloaded. This is unlike TCP, where the first N bytes of a file are available at any instant. Because there is no guarantee that the bytes of a file are received in order in BitTorrent, it can't be used for streaming video where video is played as it is received. However, if the goal is to open a file after the *entire* file has been downloaded, BitTorrent can provide better performance than TCP in many cases, because it doesn't suffer from head-of-line blocking like TCP.[2]

BitTorrent also has to deal with the problem of incentives. Every user really just wants to download their file as soon as possible. There is no incentive for a BitTorrent user—besides human goodness—to allow another user to download from them (i.e., to *upload* to another peer). In a centralized client-server model, incentives are clear: when you search on Google or watch a video on YouTube, Google shows you several ads in addition

[1] Napster would go on to have a second life as an online music store.

[2] Recall that head-of-line blocking is the situation when a single lost packet prevents delivery of subsequent packets that were received after the lost packet.

to the search results or video. If you click on one of the ads, the company being advertised pays Google a small revenue. Over millions of ads, these revenues add up. How do we create incentives in BitTorrent?

BitTorrent uses a simple tit-for-tat strategy to solve the incentive problem, which is baked into the BitTorrent program. In this strategy, each peer uploads file pieces to peers who upload the most data to this peer. On occasion, a peer will also optimistically upload data to a peer that has not uploaded too much data to it in the hope that (1) it discovers a new peer that has just joined the BitTorrent system and is interested in the same file and (2) an uncooperative peer from before has suddenly decided to be a little more generous with their upload capacity.

2.1 The rise and fall of BitTorrent

BitTorrent accounted for about a third or more of Internet traffic (by number of bytes) during its peak in the mid 2000s. BitTorrent was very popular at the time because it provided a convenient—even if oftentimes illegal—mechanism to get content such as movie videos, games, and so on. But today it occupies a much lower fraction (around 3.35%) of the Internet's traffic, especially in the developed world. What went wrong?

BitTorrent ran into the same problems as Napster: illegal file sharing and overloaded networks. While there was nothing illegal with the BitTorrent software itself, it was often used to share illegal content much to the ire of movie studios, records labels, software vendors, and so on. Further, because BitTorrent peers that were interested in the same file could be far apart geographically, BitTorrent traffic would frequently traverse the core of the Internet and consume a large fraction of the Internet's traffic. Again, this was in contrast to client-server applications, where a caching server deployed close to the client could contain network traffic originating from the client to the client's network alone. It also did not help that BitTorrent would open up multiple different TCP connections to speed up the download of files, to the detriment of other applications (e.g., SSH) that opened up only a single TCP connection.

In response to both concerns, college campuses and ISPs started blocking BitTorrent. BitTorrent adopted a less greedy network protocol called LEDBAT that would defer to TCP in times of network congestion with the end effect that it consumed less network traffic. BitTorrent users also started encrypting their traffic to prevent ISPs from detecting illegal content within this traffic. These techniques helped BitTorrent, but ultimately, the rise of (relatively) cheap and legal alternatives (e.g., Netflix, Hulu, Spotify, YouTube) made it much less attractive to use BitTorrent. BitTorrent still finds use in the developing world where legal alternatives are either not available widely or are too costly.

What did the legal alternatives do differently? Several things.

1. Their performance was much more predictable because they were backed by servers that were mostly available, as opposed to peers who could come and go as they pleased. The net result is a more predictable download (or streaming) speed for a video on Netflix when compared to downloading the same video on BitTorrent, which could take any where from hours to days to weeks depending on how popular the file is, how many peers are online, and how cooperative they are. To further improve performance, many of these video streaming services such as YouTube, Hulu, and Netflix deploy cache servers close to their clients in the same way as traditional web applications. The availability of servers close to the client reduces latency, which also has an effect on TCP throughput and contains the traffic without burdening the Internet core.

2. They were legal because they had obtained the requisite rights from the movie studios. This meant the users did not have to worry about breaking the law when downloading content.[3]

3. The user interface for these services is simpler than BitTorrent. You do not have to find a (potentially illegal) torrent file. Instead, you can just use a familiar search bar to search for videos and play them, much like searching for static content using a search engine.

[3] This problem still continues with YouTube, but YouTube has a mechanism to remove pirated content, which BitTorrent did not, because the content would have been scattered over many peers.

4. Many of these services are streaming services where the video starts playing right away without waiting for the entire file to download, unlike BitTorrent. This again significantly improves the user experience relative to BitTorrent.

5. Last, but not the least, they are cheap. Netflix today costs 10$ per month, which is a relatively modest price to pay for legal content. YouTube and Spotify have free services, where a user "pays" by watching or listening to ads.

References

[1] Napster - Wikipedia. `https://en.wikipedia.org/wiki/Napster`.

[2] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, 2003.

[3] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *SOSP*, 2007.

[4] John H Howard et al. *An overview of the Andrew File System*. Carnegie Mellon University, Information Technology Center, 1988.

[5] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-addressable Network. In *SIGCOMM*, 2001.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, 2001.

[7] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[8] Jeremy Stribling, Isaac G Councill, Jinyang Li, M Frans Kaashoek, David R Karger, Robert Morris, and Scott Shenker. OverCite: A cooperative digital research library. In *IPTPS*, 2005.

[9] Jeremy Stribling, Yair Sovran, Irene Zhang, Xavid Pretzer, Jinyang Li, M Frans Kaashoek, and Robert Morris. Flexible, Wide-Area Storage for Distributed Systems with WheelFS. In *NSDI*, 2009.

[10] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Berkeley, CA, USA, 2001.