

BitTorrent

BitTorrent is a popular P2P protocol for file distribution [Chao 2011]. In BitTorrent lingo, the collection of all peers participating in the distribution of a particular file is called a *torrent*. Peers in a torrent download equal-size *chunks* of the file from one another, with a typical chunk size of 256 KBytes. When a peer first joins a torrent, it has no chunks. Over time it accumulates more and more chunks. While it downloads chunks it also uploads chunks to other peers. Once a peer has acquired the entire file, it may (selfishly) leave the torrent, or (altruistically) remain in the torrent and continue to upload chunks to other peers. Also, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent.

Let's now take a closer look at how BitTorrent operates. Since BitTorrent is a rather complicated protocol and system, we'll only describe its most important mechanisms, sweeping some of the details under the rug; this will allow us to see the forest through the trees. Each torrent has an infrastructure node called a *tracker*. When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent. In this manner, the tracker keeps track of the peers that are participating in the torrent. A given torrent may have fewer than ten or more than a thousand peers participating at any instant of time.

As shown in Figure 2.26, when a new peer, Alice, joins the torrent, the tracker randomly selects a subset of peers (for concreteness, say 50) from the set of participating peers, and sends the IP addresses of these 50 peers to Alice. Possessing this list of peers, Alice attempts to establish concurrent TCP connections with all the peers on this list. Let's call all the peers with which Alice succeeds in establishing a TCP connection "neighboring peers." (In Figure 2.26, Alice is shown to have only three neighboring peers. Normally, she would have many more.) As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections with Alice. So a peer's neighboring peers will fluctuate over time.

At any given time, each peer will have a subset of chunks from the file, with different peers having different subsets. Periodically, Alice will ask each of her neighboring peers (over the TCP connections) for the list of the chunks they have. If Alice has L different neighbors, she will obtain L lists of chunks. With this knowledge, Alice will issue requests (again over the TCP connections) for chunks she currently does not have.

So at any given instant of time, Alice will have a subset of chunks and will know which chunks her neighbors have. With this information, Alice will have two important decisions to make. First, which chunks should she request first from her neighbors? And second, to which of her neighbors should she send requested chunks? In deciding which chunks to request, Alice uses a technique called **rarest first**. The idea is to determine, from among the chunks she does not have, the chunks that are the rarest among her neighbors (that is, the chunks that have the fewest repeated copies among her neighbors) and then request those rarest chunks first. In this manner, the rarest chunks get more quickly redistributed, aiming to (roughly) equalize the numbers of copies of each chunk in the torrent.

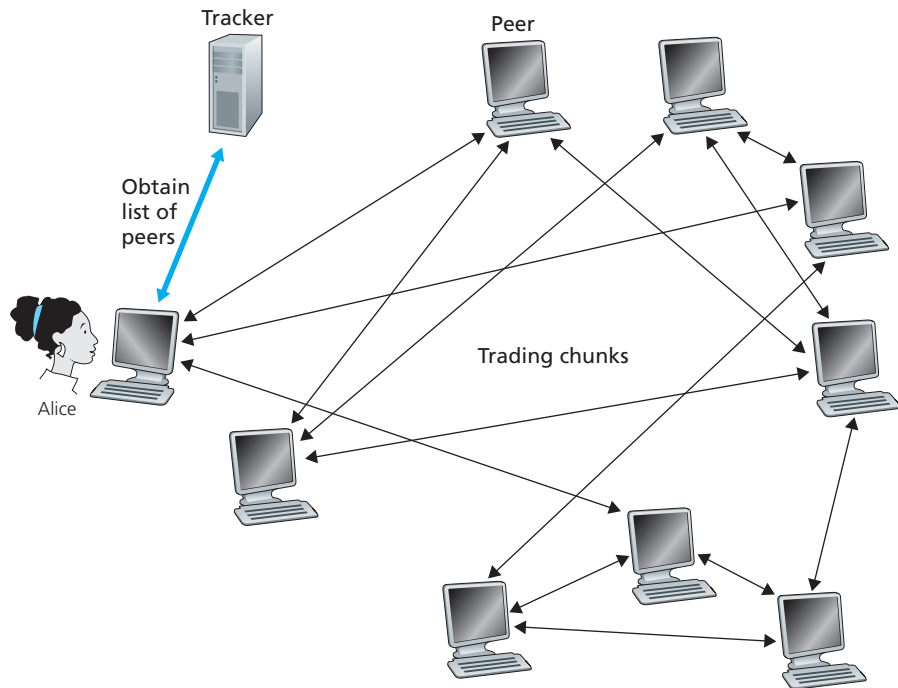


Figure 2.26 ♦ File distribution with BitTorrent

To determine which requests she responds to, BitTorrent uses a clever trading algorithm. The basic idea is that Alice gives priority to the neighbors that are currently supplying her data *at the highest rate*. Specifically, for each of her neighbors, Alice continually measures the rate at which she receives bits and determines the four peers that are feeding her bits at the highest rate. She then reciprocates by sending chunks to these same four peers. Every 10 seconds, she recalculates the rates and possibly modifies the set of four peers. In BitTorrent lingo, these four peers are said to be **unchoked**. Importantly, every 30 seconds, she also picks one additional neighbor at random and sends it chunks. Let's call the randomly chosen peer Bob. In BitTorrent lingo, Bob is said to be **optimistically unchoked**. Because Alice is sending data to Bob, she may become one of Bob's top four uploaders, in which case Bob would start to send data to Alice. If the rate at which Bob sends data to Alice is high enough, Bob could then, in turn, become one of Alice's top four uploaders. In other words, every 30 seconds, Alice will randomly choose a new trading partner and initiate trading with that partner. If the two peers are satisfied with the trading, they will put each other in their top four lists and continue trading with each other until one of the peers finds a better partner. The effect is that peers capable of uploading at compatible rates tend to find each other. The random neighbor selection also allows new peers to get

chunks, so that they can have something to trade. All other neighboring peers besides these five peers (four “top” peers and one probing peer) are “choked,” that is, they do not receive any chunks from Alice. BitTorrent has a number of interesting mechanisms that are not discussed here, including pieces (mini-chunks), pipelining, random first selection, endgame mode, and anti-snubbing [Cohen 2003].

The incentive mechanism for trading just described is often referred to as tit-for-tat [Cohen 2003]. It has been shown that this incentive scheme can be circumvented [Liogkas 2006; Locher 2006; Piatek 2007]. Nevertheless, the BitTorrent ecosystem is wildly successful, with millions of simultaneous peers actively sharing files in hundreds of thousands of torrents. If BitTorrent had been designed without tit-for-tat (or a variant), but otherwise exactly the same, BitTorrent would likely not even exist now, as the majority of the users would have been freeriders [Saroiu 2002].

Interesting variants of the BitTorrent protocol are proposed [Guo 2005; Piatek 2007]. Also, many of the P2P live streaming applications, such as PPLive and ppsream, have been inspired by BitTorrent [Hei 2007].

2.6.2 Distributed Hash Tables (DHTs)

In this section, we will consider how to implement a simple database in a P2P network. Let’s begin by describing a centralized version of this simple database, which will simply contain (key, value) pairs. For example, the keys could be social security numbers and the values could be the corresponding human names; in this case, an example key-value pair is (156-45-7081, Johnny Wu). Or the keys could be content names (e.g., names of movies, albums, and software), and the value could be the IP address at which the content is stored; in this case, an example key-value pair is (Led Zeppelin IV, 128.17.123.38). We query the database with a key. If there are one or more key-value pairs in the database that match the query key, the database returns the corresponding values. So, for example, if the database stores social security numbers and their corresponding human names, we can query with a specific social security number, and the database returns the name of the human who has that social security number. Or, if the database stores content names and their corresponding IP addresses, we can query with a specific content name, and the database returns the IP addresses that store the specific content.

Building such a database is straightforward with a client-server architecture that stores all the (key, value) pairs in one central server. So in this section, we’ll instead consider how to build a distributed, P2P version of this database that will store the (key, value) pairs over millions of peers. In the P2P system, each peer will only hold a small subset of the totality of the (key, value) pairs. We’ll allow any peer to query the distributed database with a particular key. The distributed database will then locate the peers that have the corresponding (key, value) pairs and return the key-value pairs to the querying peer. Any peer will also be allowed to insert new key-value pairs into the database. Such a distributed database is referred to as a **distributed hash table (DHT)**.