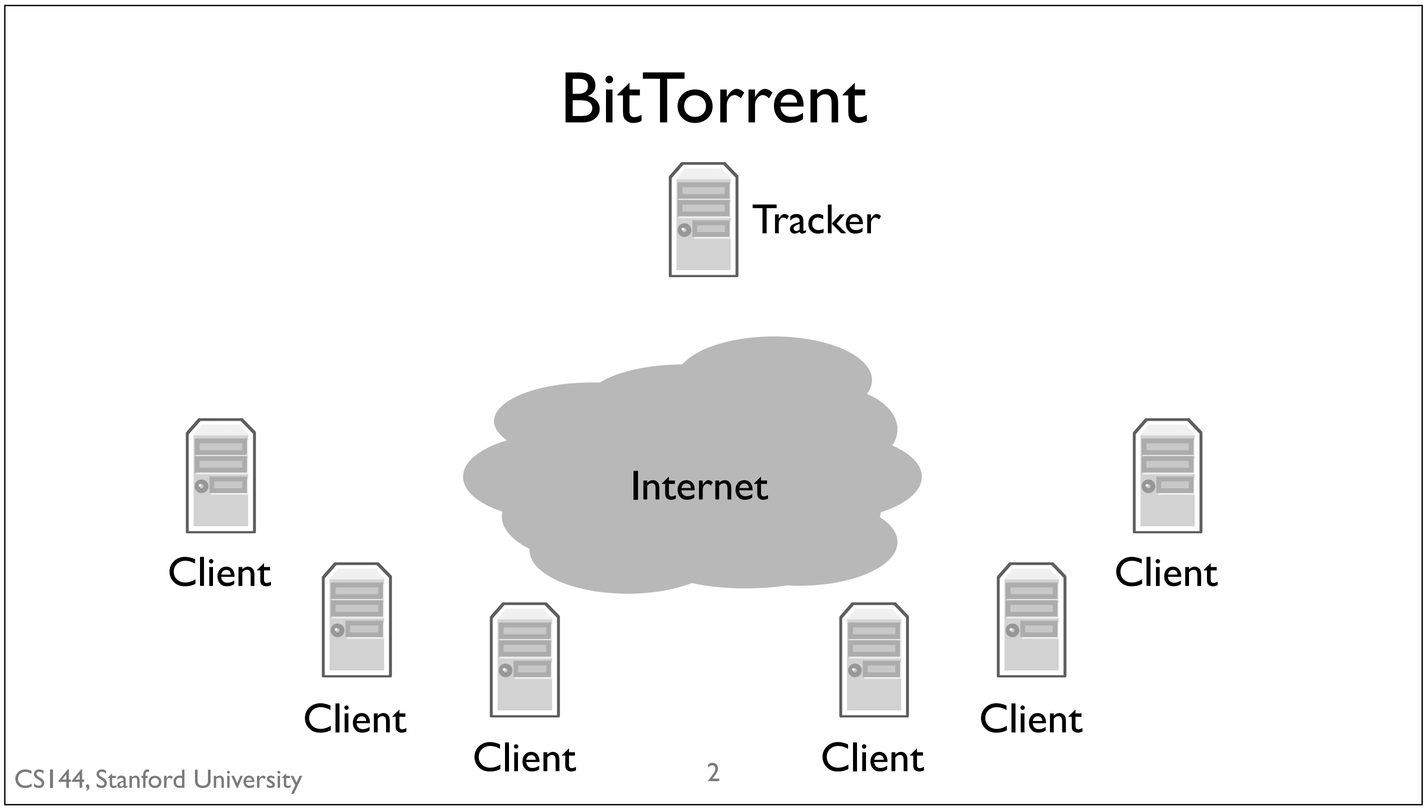
BitTorrent

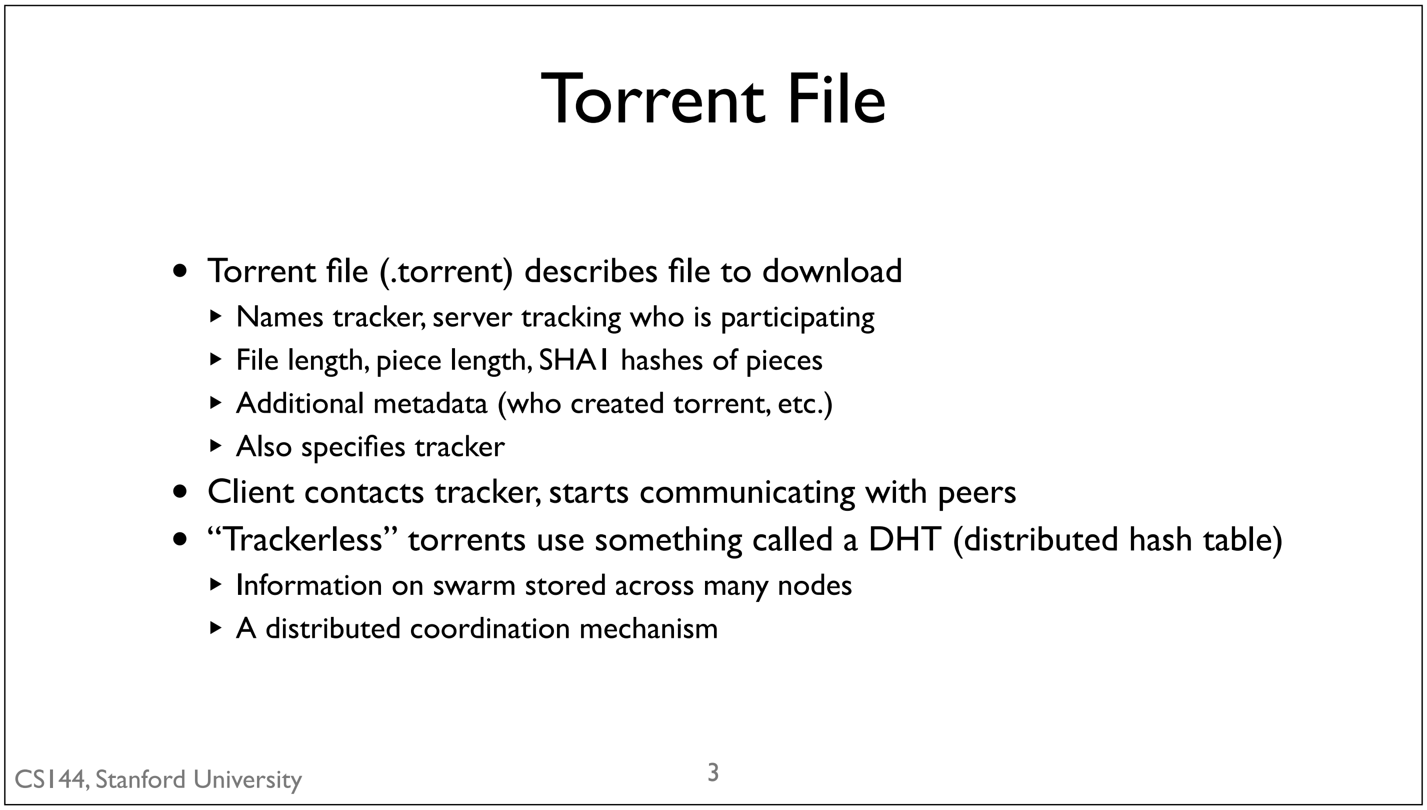
Swarms, Rarest-First and Tit-for-Tat

So let's talk about BitTorrent. It's a fascinating Internet application, with a lot of interesting algorithms and approaches. There's a reason it works so well!



BitTorrent

BitTorrent allows people to share and exchange large files. A BitTorrent client requests documents from other clients. So that a single client can request from many others in parallel, BitTorrent breaks files up into chunks of data called pieces. When a client downloads a complete piece from another client, it then tells other clients it has that piece so they can download it too. These collections of collaborating clients are called swarms. So we talk about a client joining or leaving the swarm.



Torrent File

- Torrent file (.torrent) describes file to download
  - Names tracker, server tracking who is participating
  - File length, piece length, SHA1 hashes of pieces
  - Additional metadata (who created torrent, etc.)
  - Also specifies tracker
- Client contacts tracker, starts communicating with peers
- "Trackerless" torrents use something called a DHT (distributed hash table)
  - Information on swarm stored across many nodes
  - A distributed coordination mechanism

A client joins a swarm by downloading a Torrent file that tells it information about the file, such as how big it is, the size of its pieces, and how to start contacting other clients. It used to be that a torrent would name a tracker, a computer that keeps track of what clients are part of the swarm. When a client joins the swarm, it requests a list of other clients from the tracker. It then starts contacting clients over TCP. A BitTorrent client can have on the order of 100 open TCP connections at once.

After trackers started to receive a lot of unwanted attention, in the late 2000s most clients transitioned to using trackerless torrents. These torrents contact a host that tells them how to join something called a distributed hash table, or DHT. A DHT is a way to map a hash value to a node, where the set of nodes supporting that DHT can change a lot yet you can still find the node. Rather than use a centralized table for this lookup, the mapping is actually distributed across all the participating nodes. It's basically a way for many nodes to collaboratively store some data. In this case, they're storing lists of which clients are part of a swarm.

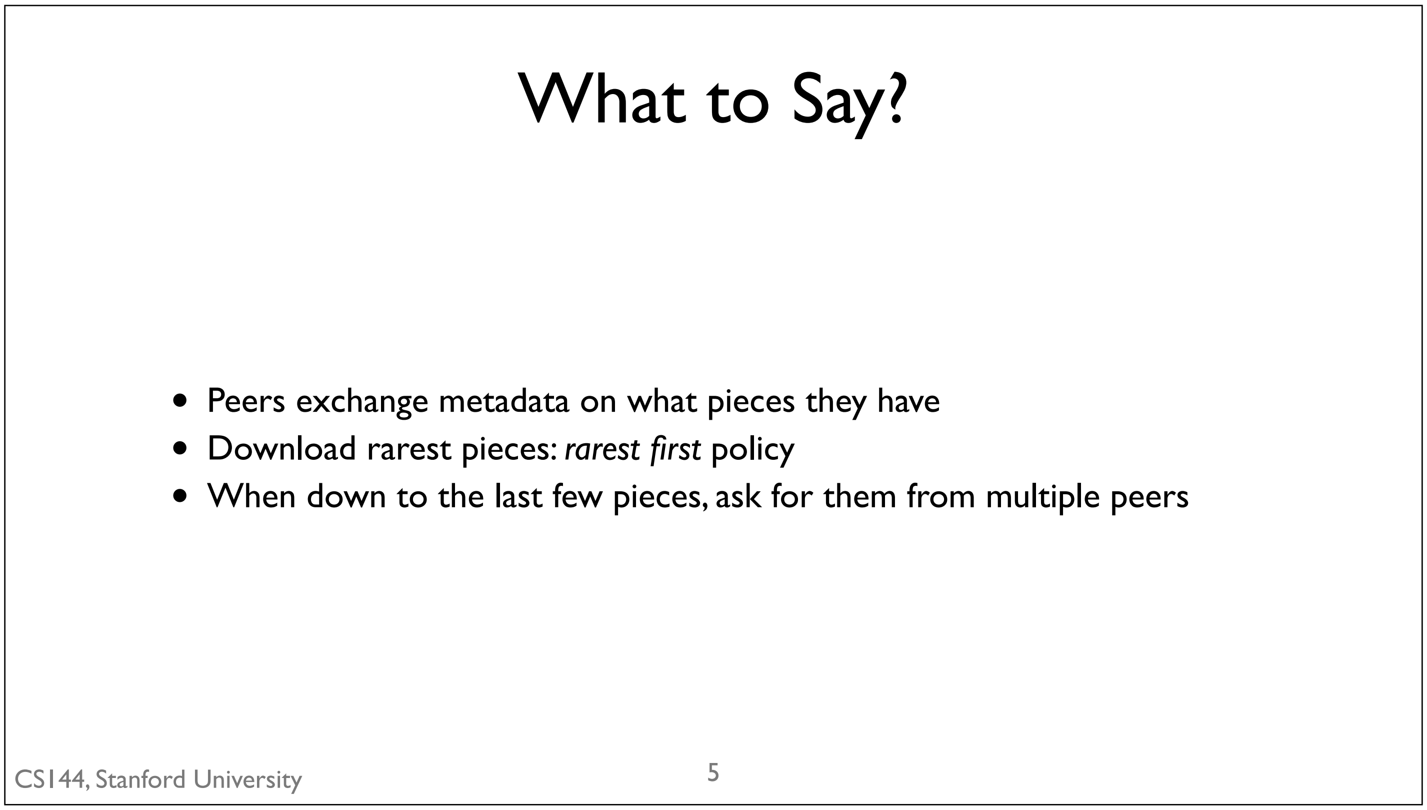
Torrent Files

- BitTorrent breaks a file up into N pieces
  - For throughput, pieces are large: 256kB-1MB
  - For latency, broken into subpieces
- Hashes of pieces in torrent provide end-to-end integrity
  - Hash computes a short summary of a piece
  - Cryptographically strong hashes: hard to create a piece of data that has a particular hash (more in security lectures
  - HBO's Rome series: blacklisting peers

BitTorrent breaks a files up into N pieces. Each piece is 256kB or larger. This size is intended to ensure a TCP stream transferring the file is long-lived enough that its congestion window can grow to a reasonable size and so support good throughput. But BitTorrent also breaks up pieces into subpieces, such that it can request parts of pieces from multiple peers and so reduce latency.

A piece is also the unit that BitTorrent uses to check integrity with. A torrent contains the SHA1 hashes of each piece. SHA1 is something called a cryptographic hash function. It's the primitive used in message authentication codes. A strong cryptographic hash function has the properties that, given a hash, it's really hard to create a piece of data that has that hash value. This means that if the torrent says that the hash of piece 5 is H, it's hard to come up with a piece that isn't piece 5 which has hash H. So you can't start replacing the pieces of the torrent and screw it up without a client noticing that the hash isn't right and retrying.

This brings up an interesting story. In 2006, HBO had a new series, Rome. There were several different torrents for it, each of which had very large swarms. But many people found their clients couldn't download the series. Looking into it, it turns out that there were a bunch of very very fast peers that many clients were connecting to and downloading from. But these peers provided pieces that didn't have the right hash. So a client would download the piece, find the hash was wrong, throw away the piece, and retry. Back then, the clients assumed that this was just an error so kept on requesting from the same peer. So many clients would just enter an unending loop of trying to download the same bad piece. The hypothesis was that this was an effort by HBO to prevent downloads. Nowadays clients can "blacklist" peers that serve up many bad pieces.



What to Say?

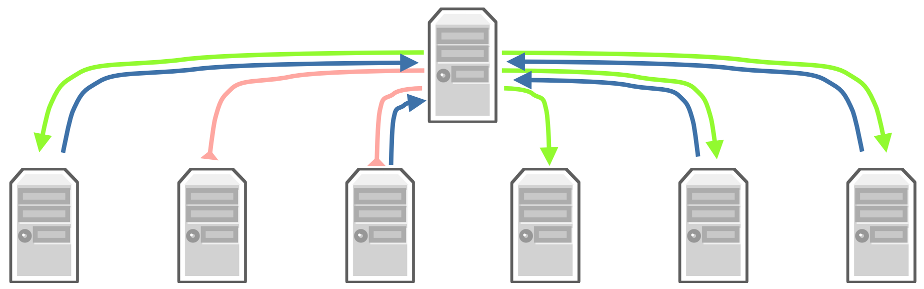
- Peers exchange metadata on what pieces they have
- Download rarest pieces: *rarest first* policy
- When down to the last few pieces, ask for them from multiple peers

BitTorrent clients, when connected, periodically exchange information on which pieces they have. A client tries to download the rarest piece among its peers first. If a single piece becomes unavailable, nobody can download the file. Also, if only a few clients have a piece, they'll become a bottleneck for downloading. This is called the rarest first policy.

The one exception to the rarest first policy is when a client reaches the end of the torrent and only needs a few more pieces. At this point, it requests for pieces from multiple peers. It does this to counter the edge case of asking for the last piece from a very slow peer. This final step means that the client might download multiple copies of subpieces and waste swarm bandwidth, but since there are often 1000 or so pieces in a swarm this cost is seen as small and worth it.

Whom To Talk To?

- Use *Tit-for-Tat* (TFT) policy: upload data to peers that give you did
- Most peers are “choked” and get no data
- Order unchoked peers by download rate, choke all but *P* best (e.g., 4, $\sqrt{C}$)
- Occassionally unchoke a random peer (might find way into *P* best)



So BitTorrent clients exchange metadata with each other to learn what pieces they have. A client starts requesting pieces from its peers. But if you sent data to every peer, you’d have lots of very slow pieces. Instead of having a hundred slow TCP flows, BitTorrent tries to have a smaller number of fast flows. You send data to peers who send you data. That way, peers who contribute can download faster. It creates an incentive to send pieces to peers.

The way this works is through choking. Most peers are choked and you send no data to them. BitTorrent measures the rate at which it is downloading from each of its peers and picks the P best of them. P is usually a small number, like 4 or the square root of the number of peers. It unchokes these P peers and sends data to them.

One problem with this algorithm is that it doesn’t explore much. There could be a really good peer out there who would send you data very fast if only you started sending some data first. So every 30 seconds or so, BitTorrent unchokes a random peer. This peer might then find its way into the P best.

BitTyrant

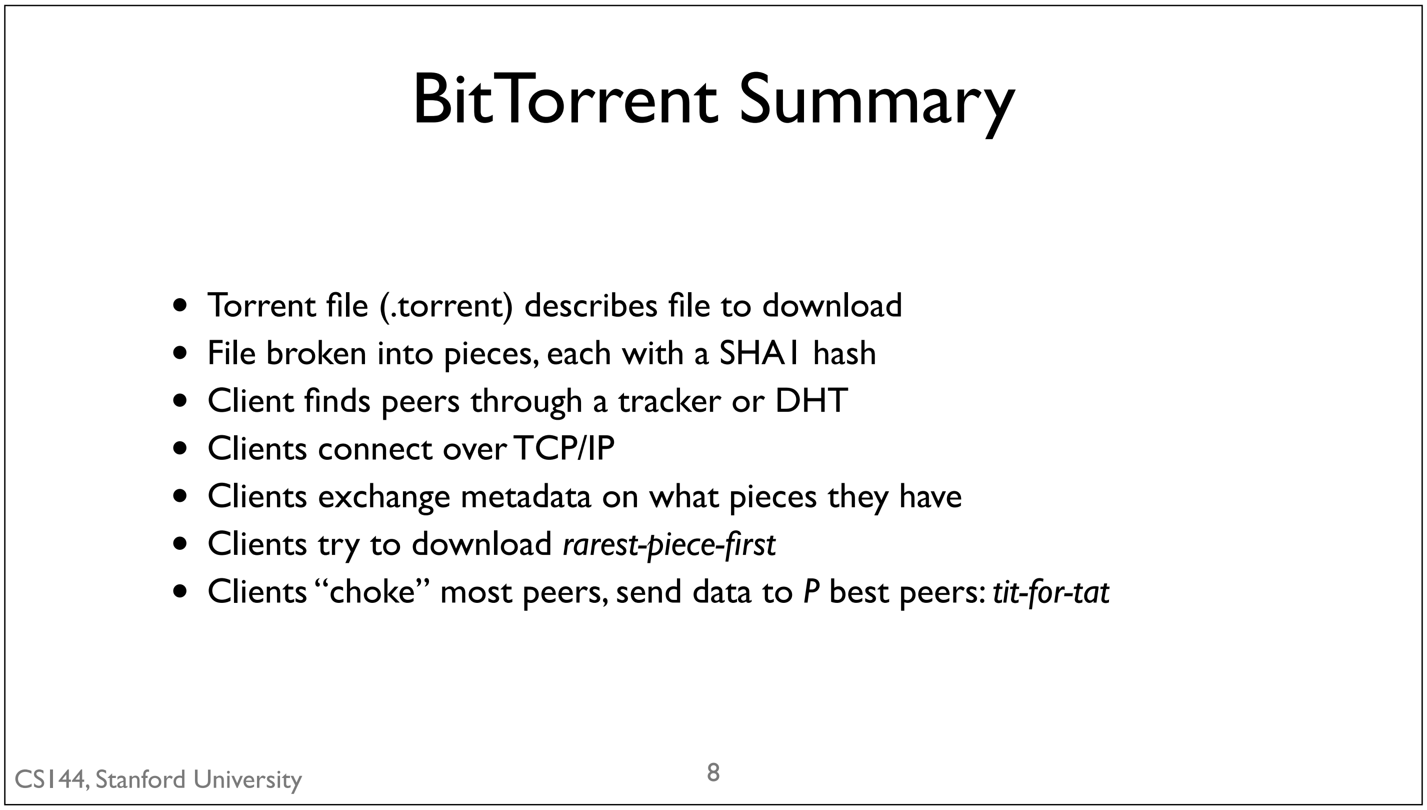
http://www.usenix.org/events/nsdi07/tech/piatek/piatek_html/bittyrant.html

- Can you game the BitTorrent Tit-for-Tat system?
- Many peers give more than they take
  - Give a peer just enough that it unchokes you
  - Convince as many peers as possible to unchoke you
  - Share capacity across more peers rather than give each peer more
- Leads to a 70% median performance gain!

The BitTorrent tit-for-tat algorithm seems pretty robust: you send data preferentially to other peers who send you data. But it's not perfect. There was a nice paper in 2007 that proposed something called BitTyrant, which selfishly tried to game the system. And it did! Using BitTyrant you could increase your BitTorrent throughput by 70%!

The basic observation is that in the standard BitTorrent, a peer tries to share its uplink capacity evenly across its unchoked peers. So if a client has P unchoked peers, then each one receives 1 over P of its uplink capacity. But once you're in this top P, you get all of this. So the trick is that you want to give a peer *just enough* to make your way into its top P, and no more. You should spend the extra capacity on trying to get into another peer's top P. So this way you try to give everyone just enough that they unchoke you, and maximize how many peers unchoke you.

It's a nice result. They also found that if everyone used BitTyrant, performance can improve slightly. But you get the most benefit if you're the only tyrant. The URL here links to the paper.



BitTorrent Summary

- Torrent file (.torrent) describes file to download
- File broken into pieces, each with a SHA1 hash
- Client finds peers through a tracker or DHT
- Clients connect over TCP/IP
- Clients exchange metadata on what pieces they have
- Clients try to download *rarest-piece-first*
- Clients "choke" most peers, send data to *P* best peers: *tit-for-tat*

CS144, Stanford University 8

So that's a basic overview of BitTorrent. Your client downloads a torrent file, for example, over HTTP. This describes the file to download and how to find peers to download from. BitTorrent breaks the file into pieces, and peers exchange these pieces. They connect over TCP/IP and exchange metadata so they know what the distribution of pieces is over their part of the swarm. A client then tries to download the rarest piece first, in order to balance availability. Clients upload data only to their top P downloaders. So most of the peers are "choked" and receive no data, and the client gives data to those who give you data using a "tit-for-tat" algorithm. To discover potentially good new peers, the client also randomly unchokes a peer periodically.