7.2.4 Content Distribution Networks

Today, many Internet video companies are distributing on-demand multi-Mbps streams to millions of users on a daily basis. YouTube, for example, with a library of hundreds of millions of videos, distributes hundreds of millions of video streams to users around the world every day [Ding 2011]. Streaming all this traffic to locations all over the world while providing continuous playout and high interactivity is clearly a challenging task.

For an Internet video company, perhaps the most straightforward approach to providing streaming video service is to build a single massive data center, store all of its videos in the data center, and stream the videos directly from the data center to clients worldwide. But there are three major problems with this approach. First, if the client is far from the data center, server-to-client packets will cross many communication links and likely pass through many ISPs, with some of the ISPs possibly located on different continents. If one of these links provides a throughput that is less than the video consumption rate, the end-to-end throughput will also be below the consumption rate, resulting in annoying freezing delays for the user. (Recall from Chapter 1 that the end-to-end throughput of a stream is governed by the throughput in the bottleneck link.) The likelihood of this happening increases as the number of links in the end-to-end path increases. A second drawback is that a popular video will likely be sent many times over the same communication links. Not only does this waste network bandwidth, but the Internet video company itself will be paying its provider ISP (connected to the data center) for sending the *same* bytes into the Internet over and over again. A third problem with this solution is that a single data center represents a single point of failure—if the data center or its links to the Internet goes down, it would not be able to distribute *any* video streams.

In order to meet the challenge of distributing massive amounts of video data to users distributed around the world, almost all major video-streaming companies make use of **Content Distribution Networks (CDNs)**. A CDN manages servers in multiple geographically distributed locations, stores copies of the videos (and other types of Web content, including documents, images, and audio) in its servers, and attempts to direct each user request to a CDN location that will provide the best user experience. The CDN may be a **private CDN**, that is, owned by the content provider itself; for example, Google's CDN distributes YouTube videos and other types of content. The CDN may alternatively be a **third-party CDN** that distributes content on behalf of multiple content providers; Akamai's CDN, for example, is a third-party CDN that distributes Netflix and Hulu content, among others. A very readable overview of modern CDNs is [Leighton 2009].

CDNs typically adopt one of two different server placement philosophies [Huang 2008]:

- **Enter Deep.** One philosophy, pioneered by Akamai, is to *enter deep* into the access networks of Internet Service Providers, by deploying server clusters in access ISPs all over the world. (Access networks are described in Section 1.3.)

CASE STUDY

GOOGLE'S NETWORK INFRASTRUCTURE

To support its vast array of cloud services—including search, gmail, calendar, YouTube video, maps, documents, and social networks—Google has deployed an extensive private network and CDN infrastructure. Google's CDN infrastructure has three tiers of server clusters:

- Eight "mega data centers," with six located in the United States and two located in Europe [Google Locations 2012], with each data center having on the order of 100,000 servers. These mega data centers are responsible for serving dynamic (and often personalized) content, including search results and gmail messages.
- About 30 "bring-home" clusters (see discussion in 7.2.4), with each cluster consisting on the order of 100–500 servers [Adhikari 2011a]. The cluster locations are distributed around the world, with each location typically near multiple tier-1 ISP PoPs. These clusters are responsible for serving static content, including YouTube videos [Adhikari 2011a].
- Many hundreds of "enter-deep" clusters (see discussion in 7.2.4), with each cluster located within an access ISP. Here a cluster typically consists of tens of servers within a single rack. These enter-deep servers perform TCP splitting (see Section 3.7) and serve static content [Chen 2011], including the static portions of Web pages that embody search results.

All of these data centers and cluster locations are networked together with Google's own private network, as part of one enormous AS (AS 15169). When a user makes a search query, often the query is first sent over the local ISP to a nearby enter-deep cache, from where the static content is retrieved; while providing the static content to the client, the nearby cache also forwards the query over Google's private network to one of the mega data centers, from where the personalized search results are retrieved. For a YouTube video, the video itself may come from one of the bring-home caches, whereas portions of the Web page surrounding the video may come from the nearby enter-deep cache, and the advertisements surrounding the video come from the data centers. In summary, except for the local ISPs, the Google cloud services are largely provided by a network infrastructure that is independent of the public Internet.

Akamai takes this approach with clusters in approximately 1,700 locations. The goal is to get close to end users, thereby improving user-perceived delay and throughput by decreasing the number of links and routers between the end user and the CDN cluster from which it receives content. Because of this highly distributed design, the task of maintaining and managing the clusters becomes challenging.

- **Bring Home.** A second design philosophy, taken by Limelight and many other CDN companies, is to *bring the ISPs home* by building large clusters at a smaller number (for example, tens) of key locations and connecting these clusters using a private high-speed network. Instead of getting inside the access ISPs, these CDNs typically place each cluster at a location that is simultaneously near the PoPs (see Section 1.3) of many tier-1 ISPs, for example, within a few miles of both AT&T and Verizon PoPs in a major city. Compared with the enter-deep design philosophy, the bring-home design typically results in lower maintenance and management overhead, possibly at the expense of higher delay and lower throughput to end users.

Once its clusters are in place, the CDN replicates content across its clusters. The CDN may not want to place a copy of every video in each cluster, since some videos are rarely viewed or are only popular in some countries. In fact, many CDNs do not push videos to their clusters but instead use a simple pull strategy: If a client requests a video from a cluster that is not storing the video, then the cluster retrieves the video (from a central repository or from another cluster) and stores a copy locally while streaming the video to the client at the same time. Similar to Internet caches (see Chapter 2), when a cluster's storage becomes full, it removes videos that are not frequently requested.

CDN Operation

Having identified the two major approaches toward deploying a CDN, let's now dive down into the nuts and bolts of how a CDN operates. When a browser in a user's host is instructed to retrieve a specific video (identified by a URL), the CDN must intercept the request so that it can (1) determine a suitable CDN server cluster for that client at that time, and (2) redirect the client's request to a server in that cluster. We'll shortly discuss how a CDN can determine a suitable cluster. But first let's examine the mechanics behind intercepting and redirecting a request.

Most CDNs take advantage of DNS to intercept and redirect requests; an interesting discussion of such a use of the DNS is [Vixie 2009]. Let's consider a simple example to illustrate how DNS is typically involved. Suppose a content provider, NetCinema, employs the third-party CDN company, KingCDN, to distribute its videos to its customers. On the NetCinema Web pages, each of its videos is assigned a URL that includes the string "video" and a unique identifier for the video itself; for example, Transformers 7 might be assigned http://video.netcinema.com/6Y7B23V. Six steps then occur, as shown in Figure 7.4:

1. The user visits the Web page at NetCinema.
2. When the user clicks on the link http://video.netcinema.com/6Y7B23V, the user's host sends a DNS query for video.netcinema.com.

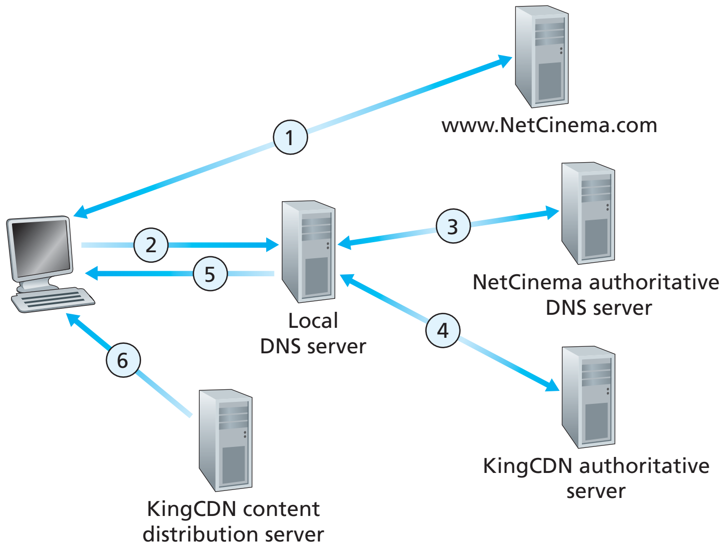

www.NetCinema.com
1
2
5
3
4
6
Local DNS server
NetCinema authoritative DNS server
KingCDN authoritative server
KingCDN content distribution server

Figure 7.4 ♦ DNS redirects a user's request to a CDN server

3. The user's Local DNS Server (LDNS) relays the DNS query to an authoritative DNS server for NetCinema, which observes the string "video" in the hostname video.netcinema.com. To "hand over" the DNS query to KingCDN, instead of returning an IP address, the NetCinema authoritative DNS server returns to the LDNS a hostname in the KingCDN's domain, for example, a1105.kingcdn.com.
4. From this point on, the DNS query enters into KingCDN's private DNS infrastructure. The user's LDNS then sends a second query, now for a1105.kingcdn.com, and KingCDN's DNS system eventually returns the IP addresses of a KingCDN content server to the LDNS. It is thus here, within the KingCDN's DNS system, that the CDN server from which the client will receive its content is specified.
5. The LDNS forwards the IP address of the content-serving CDN node to the user's host.
6. Once the client receives the IP address for a KingCDN content server, it establishes a direct TCP connection with the server at that IP address and issues an HTTP GET request for the video. If DASH is used, the server will first send to the client a manifest file with a list of URLs, one for each version of the video, and the client will dynamically select chunks from the different versions.

Cluster Selection Strategies

At the core of any CDN deployment is a **cluster selection strategy**, that is, a mechanism for dynamically directing clients to a server cluster or a data center within the CDN. As we just saw, the CDN learns the IP address of the client's LDNS server via the client's DNS lookup. After learning this IP address, the CDN needs to select an appropriate cluster based on this IP address. CDNs generally employ proprietary cluster selection strategies. We now briefly survey a number of natural approaches, each of which has its own advantages and disadvantages.

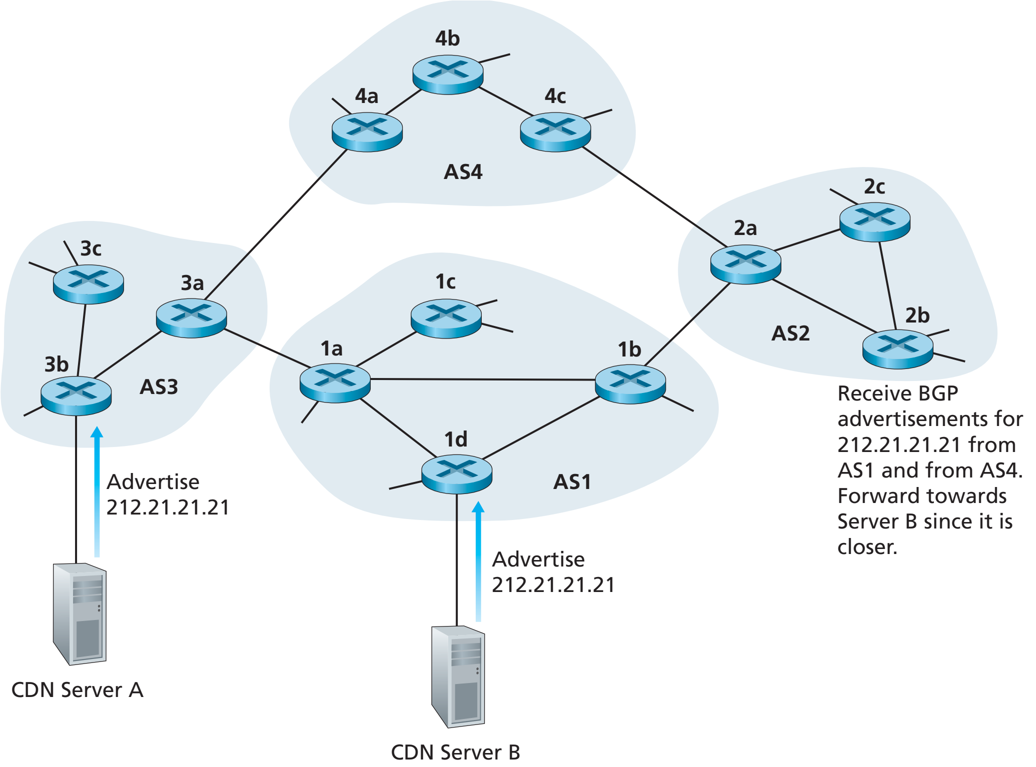
One simple strategy is to assign the client to the cluster that is **geographically closest**. Using commercial geo-location databases (such as Quova [Quova 2012] and Max-Mind [MaxMind 2012]), each LDNS IP address is mapped to a geographic location. When a DNS request is received from a particular LDNS, the CDN chooses the geographically closest cluster, that is, the cluster that is the fewest kilometers from the LDNS "as the bird flies." Such a solution can work reasonably well for a large fraction of the clients [Agarwal 2009]. However, for some clients, the solution may perform poorly, since the geographically closest cluster may not be the closest cluster along the network path. Furthermore, a problem inherent with all DNS-based approaches is that some end-users are configured to use remotely located LDNSs [Shaikh 2001; Mao 2002], in which case the LDNS location may be far from the client's location. Moreover, this simple strategy ignores the variation in delay and available bandwidth over time of Internet paths, always assigning the same cluster to a particular client.

In order to determine the best cluster for a client based on the *current* traffic conditions, CDNs can instead perform periodic **real-time measurements** of delay and loss performance between their clusters and clients. For instance, a CDN can have each of its clusters periodically send probes (for example, ping messages or DNS queries) to all of the LDNSs around the world. One drawback of this approach is that many LDNSs are configured to not respond to such probes.

An alternative to sending extraneous traffic for measuring path properties is to use the characteristics of recent and ongoing traffic between the clients and CDN servers. For instance, the delay between a client and a cluster can be estimated by examining the gap between server-to-client SYNACK and client-to-server ACK during the TCP three-way handshake. Such solutions, however, require redirecting clients to (possibly) suboptimal clusters from time to time in order to measure the properties of paths to these clusters. Although only a small number of requests need to serve as probes, the selected clients can suffer significant performance degradation when receiving content (video or otherwise) [Andrews 2002; Krishnan 2009]. Another alternative for cluster-to-client path probing is to use DNS query traffic to measure the delay between clients and clusters. Specifically, during the DNS phase (within Step 4 in Figure 7.4), the client's LDNS can be occasionally directed to different DNS authoritative servers installed at the various cluster locations, yielding DNS traffic that can then be measured between the LDNS and these cluster locations.

In this scheme, the DNS servers continue to return the optimal cluster for the client, so that delivery of videos and other Web objects does not suffer [Huang 2010].

A very different approach to matching clients with CDN servers is to use **IP anycast** [RFC 1546]. The idea behind IP anycast is to have the routers in the Internet route the client's packets to the "closest" cluster, as determined by BGP. Specifically, as shown in Figure 7.5, during the IP-anycast configuration stage, the CDN company assigns the *same* IP address to each of its clusters, and *uses standard BGP* to advertise this IP address from each of the different cluster locations. When a BGP router receives multiple route advertisements for this same IP address, it treats these advertisements as providing different paths to the same physical location (when, in fact, the advertisements are for different paths to *different* physical locations). Following standard operating procedures, the BGP router will then pick the "best" (for example, closest, as determined by AS-hop counts) route to the IP address according to its local route selection mechanism. For example, if one BGP route



Receive BGP advertisements for 212.21.21.21 from AS1 and from AS4. Forward towards Server B since it is closer.

Figure 7.5 ♦ Using IP anycast to route clients to closest CDN cluster (corresponding to one location) is only one AS hop away from the router, and all other BGP routes (corresponding to other locations) are two or more AS hops away, then the BGP router would typically choose to route packets to the location that needs to traverse only one AS (see Section 4.6). After this initial configuration phase, the CDN can do its main job of distributing content. When any client wants to see any video, the CDN's DNS returns the anycast address, no matter where the client is located. When the client sends a packet to that IP address, the packet is routed to the "closest" cluster as determined by the preconfigured forwarding tables, which were configured with BGP as just described. This approach has the advantage of finding the cluster that is closest to the client rather than the cluster that is closest to the client's LDNS. However, the IP anycast strategy again does not take into account the dynamic nature of the Internet over short time scales [Ballani 2006].

Besides network-related considerations such as delay, loss, and bandwidth performance, there are many additional important factors that go into designing a cluster selection strategy. Load on the clusters is one such factor—clients should not be directed to overloaded clusters. ISP delivery cost is another factor—the clusters may be chosen so that specific ISPs are used to carry CDN-to-client traffic, taking into account the different cost structures in the contractual relationships between ISPs and cluster operators.

7.2.5 Case Studies: Netflix, YouTube, and Kankan

We conclude our discussion of streaming stored video by taking a look at three highly successful large-scale deployments: Netflix, YouTube, and Kankan. We'll see that all these systems take very different approaches, yet employ many of the underlying principles discussed in this section.

Netflix

Generating almost 30 percent of the downstream U.S. Internet traffic in 2011, Netflix has become the leading service provider for online movies and TV shows in the United States [Sandvine 2011]. In order to rapidly deploy its large-scale service, Netflix has made extensive use of third-party cloud services and CDNs. Indeed, Netflix is an interesting example of a company deploying a large-scale online service by renting servers, bandwidth, storage, and database services from third parties while using hardly any infrastructure of its own. The following discussion is adapted from a very readable measurement study of the Netflix architecture [Adhikari 2012]. As we'll see, Netflix employs many of the techniques covered earlier in this section, including video distribution using a CDN (actually multiple CDNs) and adaptive streaming over HTTP.

Figure 7.6 shows the basic architecture of the Netflix video-streaming platform. It has four major components: the registration and payment servers, the Amazon cloud, multiple CDN providers, and clients. In its own hardware infrastructure, Netflix maintains registration and payment servers, which handle registration of new

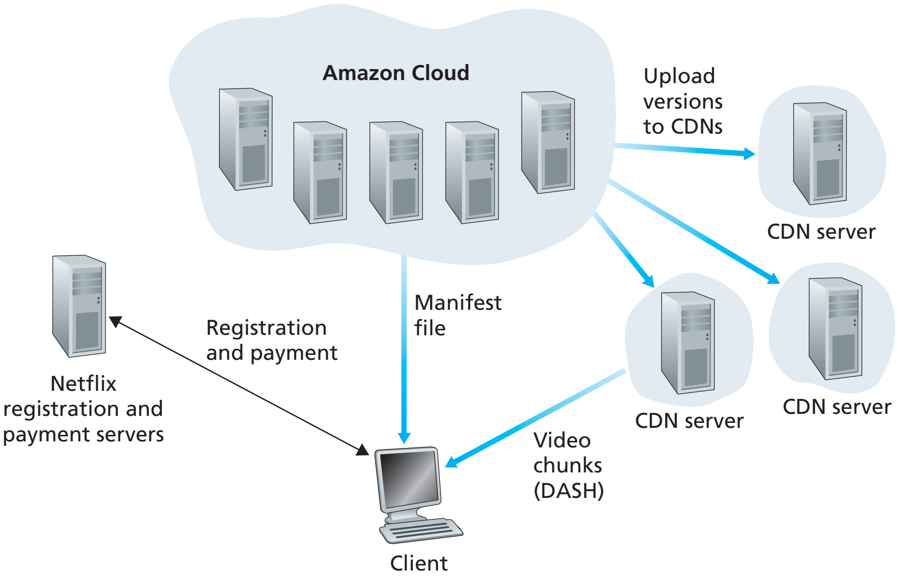


Figure 7.6 ♦ Netflix video streaming platform

accounts and capture credit-card payment information. Except for these basic functions, Netflix runs its online service by employing machines (or virtual machines) in the Amazon cloud. Some of the functions taking place in the Amazon cloud include:

- *Content ingestion.* Before Netflix can distribute a movie to its customers, it must first ingest and process the movie. Netflix receives studio master versions of movies and uploads them to hosts in the Amazon cloud.
- *Content processing.* The machines in the Amazon cloud create many different formats for each movie, suitable for a diverse array of client video players running on desktop computers, smartphones, and game consoles connected to televisions. A different version is created for each of these formats and at multiple bit rates, allowing for adaptive streaming over HTTP using DASH.
- *Uploading versions to the CDNs.* Once all of the versions of a movie have been created, the hosts in the Amazon cloud upload the versions to the CDNs.

To deliver the movies to its customers on demand, Netflix makes extensive use of CDN technology. In fact, as of this writing in 2012, Netflix employs not one but *three* third-party CDN companies at the same time—Akamai, Limelight, and Level-3.

Having described the components of the Netflix architecture, let's take a closer look at the interaction between the client and the various servers that are involved in movie delivery. The Web pages for browsing the Netflix video library are served from servers in the Amazon cloud. When the user selects a movie to "Play Now," the user's client obtains a manifest file, also from servers in the Amazon cloud. The manifest file includes a variety of information, including a ranked list of CDNs and the URLs for the different versions of the movie, which are used for DASH playback. The ranking of the CDNs is determined by Netflix, and may change from one streaming session to the next. Typically the client will select the CDN that is ranked highest in the manifest file. After the client selects a CDN, the CDN leverages DNS to redirect the client to a specific CDN server, as described in Section 7.2.4. The client and that CDN server then interact using DASH. Specifically, as described in Section 7.2.3, the client uses the byte-range header in HTTP GET request messages, to request chunks from the different versions of the movie. Netflix uses chunks that are approximately four-seconds long [Adhikari 2012]. While the chunks are being downloaded, the client measures the received throughput and runs a rate-determination algorithm to determine the quality of the next chunk to request.

Netflix embodies many of the key principles discussed earlier in this section, including adaptive streaming and CDN distribution. Netflix also nicely illustrates how a major Internet service, generating almost 30 percent of Internet traffic, can run almost entirely on a third-party cloud and third-party CDN infrastructures, using very little infrastructure of its own!

YouTube

With approximately half a billion videos in its library and half a billion video views per day [Ding 2011], YouTube is indisputably the world's largest video-sharing site. YouTube began its service in April 2005 and was acquired by Google in November 2006. Although the Google/YouTube design and protocols are proprietary, through several independent measurement efforts we can gain a basic understanding about how YouTube operates [Zink 2009; Torres 2011; Adhikari 2011a].

As with Netflix, YouTube makes extensive use of CDN technology to distribute its videos [Torres 2011]. Unlike Netflix, however, Google does not employ third-party CDNs but instead uses its own private CDN to distribute YouTube videos. Google has installed server clusters in many hundreds of different locations. From a subset of about 50 of these locations, Google distributes YouTube videos [Adhikari 2011a]. Google uses DNS to redirect a customer request to a specific cluster, as described in Section 7.2.4. Most of the time, Google's cluster selection strategy directs the client to the cluster for which the RTT between client and cluster is the lowest; however, in order to balance the load across clusters, sometimes the client is directed (via DNS) to a more distant cluster [Torres 2011]. Furthermore, if a cluster does not have the requested video, instead of fetching it from somewhere else and relaying it to the client, the cluster may return an HTTP redirect message, thereby redirecting the client to another cluster [Torres 2011].

YouTube employs HTTP streaming, as discussed in Section 7.2.2. YouTube often makes a small number of different versions available for a video, each with a different bit rate and corresponding quality level. As of 2011, YouTube does not employ adaptive streaming (such as DASH), but instead requires the user to manually select a version. In order to save bandwidth and server resources that would be wasted by repositioning or early termination, YouTube uses the HTTP byte range request to limit the flow of transmitted data after a target amount of video is prefetched.

A few million videos are uploaded to YouTube every day. Not only are YouTube videos streamed from server to client over HTTP, but YouTube uploaders also upload their videos from client to server over HTTP. YouTube processes each video it receives, converting it to a YouTube video format and creating multiple versions at different bit rates. This processing takes place entirely within Google data centers. Thus, in stark contrast to Netflix, which runs its service almost entirely on third-party infrastructures, Google runs the entire YouTube service within its own vast infrastructure of data centers, private CDN, and private global network interconnecting its data centers and CDN clusters. (See the case study on Google's network infrastructure in Section 7.2.4.)

Kankan

We just saw that for both the Netflix and YouTube services, servers operated by CDNs (either third-party or private CDNs) stream videos to clients. Netflix and YouTube not only have to pay for the server hardware (either directly through ownership or indirectly through rent), but also for the bandwidth the servers use to distribute the videos. Given the scale of these services and the amount of bandwidth they are consuming, such a "client-server" deployment is extremely costly.

We conclude this section by describing an entirely different approach for providing video on demand over the Internet at a large scale—one that allows the service provider to significantly reduce its infrastructure and bandwidth costs. As you might suspect, this approach uses P2P delivery instead of client-server (via CDNs) delivery. P2P video delivery is used with great success by several companies in China, including Kankan (owned and operated by Xunlei), PPTV (formerly PPLive), and PPs (formerly PPstream). Kankan, currently the leading P2P-based video-on-demand provider in China, has over 20 million unique users viewing its videos every month.

At a high level, P2P video streaming is very similar to BitTorrent file downloading (discussed in Chapter 2). When a peer wants to see a video, it contacts a tracker (which may be centralized or peer-based using a DHT) to discover other peers in the system that have a copy of that video. This peer then requests chunks of the video file in parallel from these other peers that have the file. Different from downloading with BitTorrent, however, requests are preferentially made for chunks that are to be played back in the near future in order to ensure continuous playback.