File Downloads with BitTorrent

Distributed Load

When a Linux distribution starts to become popular, and the number of users starts to increase, the demand for downloading can be expensive. To meet the needs of the users, the distributors need faster download capabilities. This can cause a financial problem that could end the project early as the bandwidth fees start to rise, at least for smaller distributors.

BY CARSTEN SCHNOBER

Popularity is a good thing, but it does require enormous amounts of bandwidth to support the hordes of data hungry users downloading the latest version from the distributor's servers. Data transfer can be an issue – if distributors fail to distribute the load.

Users wanting to download a large file, such as a CD image, without a flat-rate account will need a lot of patience, and a substantial amount of money, if the download freezes despite a nominally quick Internet connection. Developers who are really interested in distributing their software to the Community will want to avoid this scenario.

The Theory...

The BitTorrent concept can help both sides. People who need to host large amounts of data – such as one or multiple CD images – are the first "seeds" in BitTorrent speak. The files are split up into smaller packages. After downloading the first package, the BitTorrent downloader sets itself up as a "peer" and offers that package to other machines. This is something like the role played by a **mirror server**, although the machine only has part of the complete download.

The next user wanting to download the file can access either of these sources – the seed or the peer – to download the packages, thus becoming a peer itself. After collecting all the required packages, and provided you do not terminate the BitTorrent client at this point, your machine automatically becomes a server, offering the complete download without needing to download anything itself. In other words, your machine becomes a seed.

This design allows the total bandwidth to be distributed across the whole group of users. The more people who are interested in a file, the more peers there will be. When a sufficiently large pool of peers and seeds has been established, the originator can forget bandwidth issues on the original seed, while "customers" are happy about the quick download speed. The number of peers tends to be driven by the amount of interest in an offering; thus the primary source should not experience difficulty, even though there might be a restricted number of peers.

A so-called tracker coordinates the whole process. A tracker is a server that talks to peers, asking them what pack-

GLOSSARY

Mirror server: *To offload some of the work from one server, mirror servers provide the same content at a different location.*

Python: *A programming language for Linux and other Unix-based systems, Windows, OS/2, and Mac OS. As Python is a scripting language, it does not require a compiler to translate the program code into a machine-readable format, but it does need the Python interpreter to run the program.*

Hash: *A sequence of numbers generated from a file by an irreversible mathematical algorithm. The same input will always produce the same hash code and different input will always produce different hashes. Thus, a file can be uniquely identified by inspecting its hash code.*

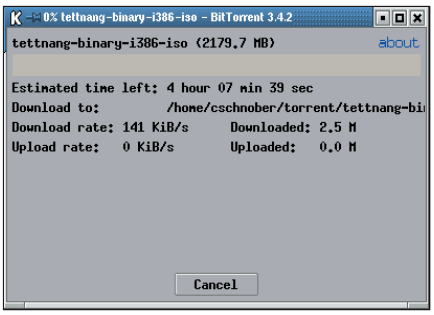

Figure 1: BitTorrent allows fast download speeds.

ages they have downloaded and are now in a position to offer. Whenever a new peer contacts the tracker, it receives a list of active peers and seeds where it can start to download.

In order to find the appropriate tracker, a Torrent file containing critical information such as the tracker's Internet address, the available files and their sizes, and a **Hash** code for the purpose of unique identification, exists for each offering. As the tracker does not support downloading itself, it needs a permanent Internet connection, but very little bandwidth.

...and Practice

A user needs the Torrent file for a specific offering in order to start downloading. These files are available on Web servers distributed throughout the Internet, and easily locatable via search engines or links. Let's suppose you are looking for the Fedora Linux distribution, and you query your favorite search engine for *Fedora* and *Torrent*. This should quickly take you to the right address, but if not, you should check the BitTorrent homepage [1] for more links.

Debian Sarge, or later, includes a ready-to-run BitTorrent program package, which can be installed by entering the *apt-get install bittorrent* command. Users of other distributions can type *tar -xzf BitTorrent-3.4.2.tar.gz* to unpack a downloaded archive into their home directories.

The BitTorrent client was written in the **Python** programming language and thus requires other resources. Besides the basic Python package, the GUI client needs the *wxWidgets* add-in for Python to handle window-dressing. Suse calls this package *python-wxGTK*, and Mandrake refers to it as *wxPythonGTK*.

Next, save the Torrent file on your hard disk and run the GUI-based BitTorrent-Client in the new BitTorrent-3.4.2 directory to start the download:

```
python btdownloadgui.py ➥
tettnang-binary-i386-iso.torrent
```

This example downloads the "Tettnang" (*Core 2*) Fedora edition (see Figure 1). Debian users can type instead:

```
btdownloadgui tettnang-binary-➥
i386-iso.torrent
```

Point to the desired target directory to start the download.

Besides the GUI-based client, BitTorrent also supports text-based downloads. Use *btdownloadcurses.py* instead of *btdownloadgui.py* to download in a text window, or run the command line program, *btdownloadheadless.py*.

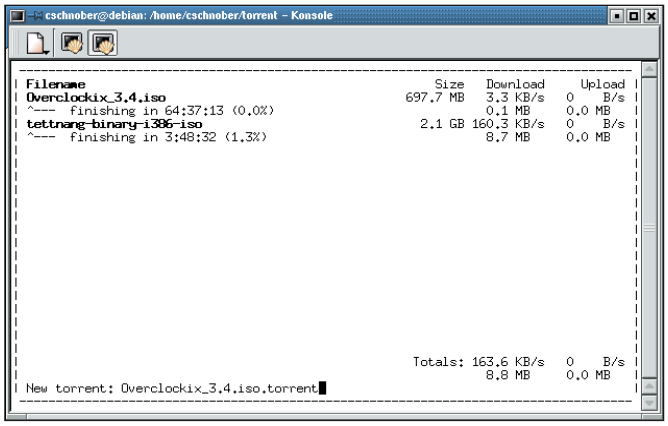

Figure 2: Multiple downloads in parallel in a text window.

The following syntax launches multiple parallel downloads:

```
python btlaunchmanycurses.py ➥
~/torrents
```

This example uses all the Torrent files in the *~/torrents/* directory, and launches the downloads in a single text window (see Figure 2). There is no GUI-support for multiple downloads, although you can run multiple clients at the same time without any trouble.

The *btshowmetainfo.py* tool provides you with additional information on the content of a Torrent offering. When calling the script, simply specify the name of a Torrent file to view the names and sizes of the files available in that file:

```
python btshowmetainfo.py ➥
tettnang-binary-i386-iso.torrent
```

Bartering

Torrent offerings that become so widespread as to have a free peer or seed for each download request, allow requesting clients to achieve very efficient download speeds. However, if a peer receives several download requests at the same time, it needs a way of deciding which request to handle first. The peer will always select the candidate that offers the most packages in exchange. Thus, the first package for a new peer might take a while to load, but the download rate will increase with every additional package it has on offer itself.

This technique prevents clients from downloading without offering anything in exchange, which is a good thing as it would effectively prevent load balancing. Things start to become more complex if you try to download from behind a firewall or router that prevents you from contacting the outside world. You will be unable to offer package downloads yourself, and that means having to live with low download speeds, if there are more requests than offers.

If you are in a position to configure your firewall, you can resolve this issue by allowing incoming connections to TCP ports 6881-6889 for BitTorrent. Otherwise, peers that support external access will not get to know their hidden colleagues until they receive a request from them, as peers automatically communicate their offerings when they talk. So clients hidden behind firewalls may still receive a positive response to their download requests, although they will remain less well-known and thus not profit as quickly from the download speeds more open peers can expect. ■

INFO

[1] BitTorrent: *http://bitconjurer.org/BitTorrent/*