Leveraging Altruistic Peers to Reduce the Bandwidth Costs of Free Content Downloads

Cameron Dale
School of Computing Science
Simon Fraser University
Burnaby, British Columbia, Canada
Email: camerond@cs.sfu.ca

Jiangchuan Liu
School of Computing Science
Simon Fraser University
Burnaby, British Columbia, Canada
Email: jcliu@cs.sfu.ca

ABSTRACT

We introduce the opportunity many free content distributors have to reduce their bandwidth costs using a peer-to-peer download mechanism. Our model for this peer-to-peer distribution is simple to implement, backwards-compatible with the existing infrastructure, and can be deployed incrementally. It makes wide use of a customized distributed hash table (DHT) to facilitate the finding of peers and the efficient downloading of files. We also present the example solution we created for one of these distributors, the Debian project, which is already seeing some use.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Design, Performance, Reliability

Keywords

BitTorrent, P2P

1. INTRODUCTION

There are a large number of free content distributors using custom package distribution systems over the Internet to distribute content to their users: e.g. most Linux distributions (Debian, Red Hat, Gentoo), Cygwin, CPAN, etc. These distributors have developed many different methods for this distribution, but they almost exclusively use a client-server model to satisfy user requests. The contents' size, popularity and the number of users results in a large number of requests, which usually requires a complicated network of mirrors to handle. Due to the free nature of this content, many users are willing to contribute upload bandwidth to this distribution, but currently have no way to do this.

There are many peer-to-peer implementations available today, but none is very well suited to this specific problem. Many distributors make their content available in some form using BitTorrent [1], but it is not ideally suited to an application such as this one. BitTorrent is designed for large static collections of files, in which all peers are interested in most or all of the content. But the packages in this case are small, constantly being updated, and users are only interested in a small subset of them. Other proposed implementations suffer from some of the same, and other, problems and so none has been well-used to address this problem.

We present a new peer-to-peer distribution model to meet these demands, based on other successful peer-to-peer protocols such as Distributed Hash Tables and BitTorrent. The model relies on the pre-existence of cryptographic hashes of the content, which uniquely identify it for a request from other peers. If the peer-to-peer download fails, then the original request to the server is used as a fallback to prevent dissatisfied users. The peer can then share this new content with others through the P2P system. This system would efficiently satisfy the demands of a large number of users, and should significantly reduce the currently substantial bandwidth costs of hosting this content.

We also present a sample implementation based on the Debian package distribution system. This implementation is simple and makes use of well-known technologies, and so serves as an example for other free content distributors of the opportunity that can be easily met with such a system.

2. PROBLEM

There are a large number of groups using the Internet to distribute their free content. This content is made available from a free download site, which usually requires a network of mirrors to support the large number of requests. This content almost always supports some kind of file verification using cryptographic hashes to verify the download's accuracy or authenticity.

Most Linux distributions use a software package management system that fetches packages from a network of mirrors. Debian-based distributions use the `apt` program, which downloads Debian packages from one of many mirrors. RPM-based distributions use `yum`, and Gentoo uses `portage`, which both operate in a similar way. Other free software distributors also use a similar system: CPAN distributes files containing software packages for the PERL programming language, using SOAP RPC requests to find and download files; Cygwin provides many of the standard Unix/Linux tools in a Windows environment, using a package management tool that requests `tar` files from websites; two software distribution systems exist for Mac OSX, fink and MacPorts, that also retrieve packages in this way. Also, some systems use direct web downloads, but with a hash verification file also available. These hash files usually have the same file name, but with an added extension identifying the hash used (e.g. `.md5` for the MD5 hash). Finally, there

SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.
ACM 978-1-60558-175-0/08/08.

are other programs that make use of cryptographic hashing to identify files: e.g. Git is a version control system in which all files and changes are identified by their SHA1 hash.

These systems all share some commonalities: the content is avaliable for anyone to freely download, the content is divided into distinct units (packages), users are typically not interested in downloading all of the content available, and hashes of the content are available before the download is attempted. We also expect that there are many users of these systems that are motivated by altruism to want to help out with this distribution. This is common in these systems due to the free nature of the content being delivered.

3. SOLUTION

This situation presents a clear opportunity to use a new peer-to-peer file sharing protocol. A downloading peer can lookup the package hash using the protocol and, if it is found, download the file from those peers and verify the content. If the package hash can not be found, the peer will fallback to downloading from the original content location, and once complete will announce to other peers indicating that it now has the content. The original servers or mirrors thus act as *seeds* for the P2P system, without any modification to them. Users are satisfied even when there are no peers, allowing the system to be deployed incrementally.

This functionality could be directly integrated into the package management software, although this would be difficult as the protocol should be running at all times, whereas the package software typically only runs until the download request is complete. Alternatively, it it possible to implement the P2P aspect as a caching proxy. The proxy will get uncached requests first from the P2P system, and then fallback to the normal request from a server.

The sparse interest in a large number of packages undergoing constant updating is well suited to the functionality provided by a Distributed Hash Table (DHT). DHTs require unique keys to store and retrieve strings of data, which the cryptographic hashes used by the package management systems are perfect for. The stored and retrieved strings can then be pointers to the peers that have the content package that hashes to that key, as well as any other information needed by the protocol to facilitate the download.

To download larger files efficiently from a number of peers, the file needs to be broken up into pieces, each with its own hash. This method, inspired by BitTorrent, makes it very easy to parallelize the downloading process and maximize the download speed. Since the package management systems only make available a hash of the entire content, the piece hashes will have to be found through the P2P protocol.

For files smaller than the piece size, no piece hashes are necessary. For medium size files of only a few pieces, the piece hashes are short enough to be stored in the DHT with the pointer to the peer to download from. For large files of 10's of pieces, the piece hashes are generally too long to store with the pointer to download peers. Instead, the peers will store the piece hashes for large files separately in the DHT using as a key the hash of the piece hashes, and include this key in their stored value for the hash of the file. Peers downloading these large files can then retrieve the piece hashes using a second DHT request. For very large files for which the piece hashes are too large to store at all in the DHT, a request to the peer for the hash (using the same method as file downloads) should return the piece hashes.

4. SAMPLE IMPLEMENTATION

To demonstrate the ease and effectiveness of this solution we have created a sample implementation called `apt-p2p` which interacts with the `apt` tool found in most Debian-based Linux distributions. `apt` uses SHA1 hashes to verify most downloaded files, including the large index files that contains detailed information such as the hashes of the individual packages. Since all requests from `apt` are in the form of HTTP downloads from a server, the implementation takes the form of a caching HTTP proxy.

The DHT used is based on Khashmir, which is an implementation of Kademlia [2] used by most of the existing BitTorrent clients to implement trackerless operation. It has been modified to better support the retrieval of multiple values (peers) per key, and to improve the response time so as to satisfy users' demands for speed. The values stored in the DHT are all bencoded dictionaries similar to torrent files. Peers store their download location (IP address and TCP port), as well as the piece strings (or hashes of piece strings) as described in the previous section. Downloading is accomplished by simple HTTP requests for the hash of the file, which is sent to the peers identified in a DHT lookup to have the desired file. All peers support HTTP/1.1, both as servers and clients, which allows for pipelining of multiple requests to a peer, and the requesting of smaller pieces of a large file using the Range request header.

We have chosen a piece size of 512 kB, which means that 17,515 (79%) of the almost 23,000 Debian packages will not require piece information as they are smaller than a single piece. There are 3054 packages (13%) that will require 2 to 4 pieces, for which the piece hashes are stored directly with the peer download information in the DHT. There are 1667 packages (7%) that will require 5 to 70 pieces, and so use a separate lookup in the DHT for the piece hashes. Finally, there are only 62 packages (0.3%) that require more than 70 pieces, and so need a separate request to a peer for the piece hashes.

We have deployed the DHT part of this implementation on PlanetLab to test it's speed and effectiveness at delivering results. We find that, even on the overloaded PlanetLab network, the responses to lookups of keys take less than 10 seconds on average. Since each package download requires a lookup before it can proceed, it may take at least 10 seconds to process a request. However, due to `apt` pipelining up to 10 requests at a time to our proxy, the user will see an average response time (after a short startup) of under a second, which should be responsive enough to satisfy most users.

This sample implementation has also been made available to all Debian users as an installable software package. Once it becomes more widely used, we plan to collect more usage information on the system through crawls and data analysis.

5. REFERENCES

[1] B. Cohen. Incentives build robustness in BitTorrent, May 2003.

[2] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. *Peer-To-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8*, 2002.