# cP2Pc: Integrating P2P networks.

Ihor Kuz, Maarten van Steen

In the world of P2P file-sharing things are moving fast. New networks based on new technologies keep appearing, while old ones are slow to disappear. Not only is there growth with regards to the number of networks and the underlying technologies used, but there is also growth in the application of these technologies. Whereas the original growth around P2P networks was based on illegal sharing of content, new (and future) applications tend to focus on legitimate distribution of content. This includes distribution of free software, distribution of Web content, implementation of streaming radio, etc. Nevertheless the basic model remains the same - distribution or sharing of content.

The problem with all the various networks is that each creates its own separate file space. There is no interoperability between networks, which results in no content being shared between networks. As a consequence, a file made available on one network can be found and accessed using programs that are tailored only to that specific network. To access multiple networks, it is therefore necessary to run multiple client programs. For example, a user who wants to publish a file on Freenet, MNet and Gnutella must start three separate programs and publish the file three times. Likewise a user looking for particular content may have to search multiple networks—and access each with a different client—before the content is found.

This is an undesirable situation that puts an unnecessary burden on end users. Users are generally not interested on which network they share files, but it should preferably be only one.

An approach to solving this problem is the creation of an *ÜberClient* as suggested by Freenet's Brandon Wiley [6]. An ÜberClient is a client application that provides a single user interface but connects to multiple P2P networks. In this article we present such an ¨ÜberClient. More importantly, however, we present a unifying API to P2P file-sharing networks on which this client is based. This API adds modularity to the ÜberClient by separating the user interface aspect from the networking aspect. We developed the API and its implementation in the *cP2Pc* project (pronounced as "copy to PC"), of which the source code is publicly available under an LGPL license and can be found at http://sourceforge.net/projects/cp2pc.

## P2P File-sharing Networks

In order to develop an ÜberClient we need to integrate the functionality of various existing P2P networks at the client level. This is possible only if the semantics of the various networks are sufficiently similar.

In the *cP2Pc* project we've examined a number of file-sharing networks and have distilled a model of the basic semantics of these networks. Not surprisingly, the basic unit of data in this model is the file. A file-sharing network supports four operations: Publish, Unpublish, Download, and Search. The Publish operation makes a file available to other users of the network. The Unpublish operation makes a previously available file unavailable. The Download operation retrieves a published file. The Search operation searches for published files which match a given criteria.

Table 1 provides an overview of how these basic functions are implemented in a number of existing file-sharing networks. We describe the networks briefly in the sidebar.

## A Unifying API

The *cP2Pc* API is a file-sharing API based on the above model. The API defines functions for publishing, unpublishing and downloading a file. It also defines a set of functions that can be used to search for files with specific characteristics.[1] In contrast to the user interfaces of many file-sharing applications, there is no functionality for automatically publishing all the files in a given directory. Each file must be separately published. Similar functionality can, however, be achieved by monitoring a directory and publishing or unpublishing files as they are added or removed.

The API also introduces the concept of *collections*. A collection is a group of related files that can be published, unpublished, searched for and downloaded separately or as a single unit. Examples of files that may be published as a collection include: tracks on an album, chapters in a book, files in a software distribution, etc.

The functions provided by the *cP2Pc* API are asynchronous, that is they return immediately while results (as well as progress reports) are returned via callbacks. This is necessary to create responsive clients, especially if a client must use multiple networks simultaneously. Specific file-sharing network implementations of this API are called components. Because some file-sharing networks do not support all features of the model, it is possible that in some component implementations some of the functions do not actually do anything (i.e., they have null implementations).

Currently we have built two *cP2Pc* network components, one for the GDN network and one for Gnutella. The Gnutella component implements all the *cP2Pc* file (publish, unpublish, download) and search functionality. It does not support functions dealing with collections. The GDN component, on the other hand, imple-

[1] The search API is directly based on Tristero's search API.

Table 1: A comparison of four file-sharing P2P networks

| **Network** | **Operation** | **Description** |
|---|---|---|
| Gnutella | Publish | Local files are made available and can be downloaded from the client using HTTP. |
| | Unpublish | Local files are no longer available. |
| | Download | Files are downloaded using HTTP. |
| | Search | Search queries are broadcast out onto the network. When a query arrives at a node, the node checks whether it has any files that match the query. |
| Freenet | Publish | Files are inserted into the network. Copies of files are placed on multiple nodes. |
| | Unpublish | Cannot explicitly unpublish a file. Files that are rarely accessed "disappear." |
| | Download | Given a file identifier a request is sent out onto the network. If a file is found it is returned (again through the network, not directly). |
| | Search | Can only search for a file given its identifier. There are external search protocols written for Freenet. |
| GDN | Publish | Files are placed in an object, which may be replicated. |
| | Unpublish | Files are removed from the object. The object may be destroyed if it is no longer needed. |
| | Download | Files are retrieved from the object. |
| | Search | No search functionality. |
| CFS | Publish | Files are created in the file system using regular system calls. |
| | Unpublish | Read only, so files cannot be deleted. |
| | Download | Files are opened and read using regular system calls. |
| | Search | Files can be found by browsing the file system. No attribute based search. |

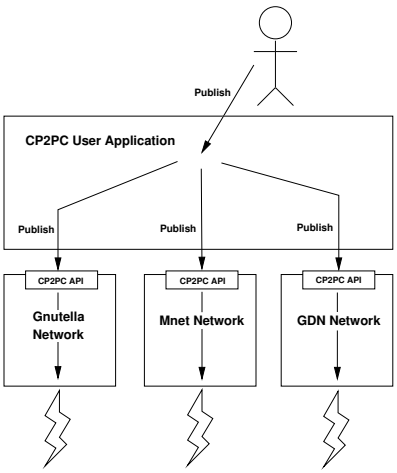


Figure 1: The organization of *cP2Pc*.

ments all the *cP2Pc* file and collection functionality. It does not, however, support the search functionality.

## An ÜberClient Architecture

An ÜberClient is a client that allows a user to access multiple networks at once. For a file-sharing application, this means that a user can publish a single file on multiple networks, or perform a search on multiple networks in a single action.

As mentioned earlier, the motivation for designing and implementing the *cP2Pc* API is to separate the ÜberClient application's interface implementation from the network-specific implementation. In this way, a component that implements the *cP2Pc* API can be used by any client that makes use of the API. Likewise, a client that is written to use the API can make use of any file-sharing network that has a *cP2Pc* component.

Figure 1 shows this relationship between the ÜberClient application logic and the *cP2Pc* network components. The application part provides the user with a (graphical) interface to the *cP2Pc* API functionality. When a user performs an action on the interface, the application invokes the appropriate functions on (all or some of) the underlying components. Each component performs the required function and returns results to the client (through a callback). The client processes results and presents them to the user.

This figure also shows that the concerns of user interface and P2P network architecture are clearly separated. With this architecture it is possible for the user interface and the network components run on separate computers (communicating through XML-RPC, for example) as shown in Figure 2(a). Given such a separation, it is also possible that multiple interfaces make use of the same *cP2Pc* network

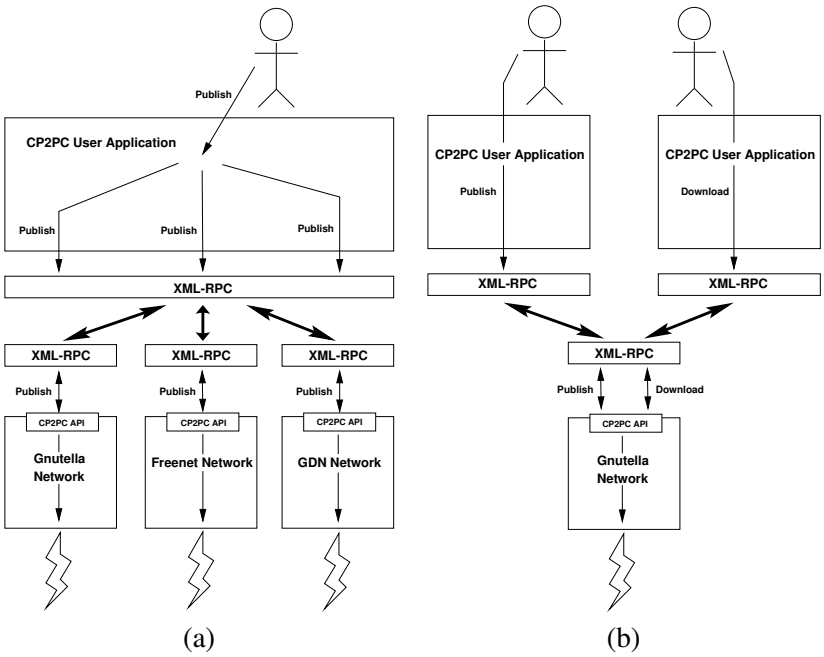


Figure 2: (a) Remote access; (b) Multiple interfaces.

component as shown in Figure 2(b).

## The cP2Pc client

Besides implementing a number of cP2Pc network components, we have also implemented a *cP2Pc* ÜberClient GUI. Currently the GUI uses the Gnutella and GDN components described above.

Figure 3 shows screenshots of the ÜberClient in action. It shows the publish screen and the search screen. Note, in particular, that a user can choose which networks to publish or search on. By default all networks are selected.

Note that GDN and Gnutella networks are technically very different. For example, publishing a file on GDN requires the file to be uploaded to a remote object, while a file published on Gnutella remains on the local machine. All of these differences are hidden by the API and are not visible to the client or the user.

The software is written in Java and runs on the major Unix platforms. We have not tested Windows compatibility. As part of the software we also provide an infrastructure that simplifies the writing of compatible network components. It is our desire that many more compatible back-ends be created (either by us or other developers) to broaden the scope of integration made possible by cP2Pc.

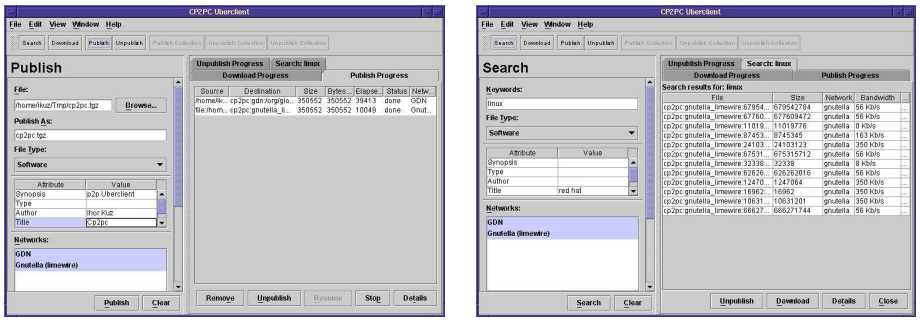


Figure 3: A screenshot for (a) publishing and (b) searching for data.

## Future directions

Although we have created a client for integrating pure file-sharing networks the file-sharing model can be applied to many different kinds of applications where data is distributed. For example the API could be use to integrate various news-distribution networks. It could also be used to create clients for file storage networks (distributed file sharing). All that would be required would be appropriate back-ends and a single front end.

An interesting issue to look into further is whether the *cP2Pc* API could be used to create unifying gateways, as also suggested by Wiley [6]. It is likely that the *cP2Pc* API as it currently stands is not sufficient for this task. However, the API would probably require only minor modifications to make it appropriate for gateway functionality.

## SIDEBAR: Some P2P Networks

Gnutella [5] is a decentralized P2P file-sharing network. It allows local files to be shared with other users and also provides a decentralized mechanism to search for files shared by other users. When performing a search, query messages are broadcast to all peers within a maximum path radius (determined by a message's time-to-live value). A node receiving an incoming search message checks if any of its shared files can satisfy the search query. If so, information about that file is returned back to the originator of the query. Files are downloaded directly from the peer hosting it using HTTP.

Freenet [3] is a P2P network which aims to provide a way to publish and obtain information on the Internet without fear of censorship. In Freenet files are stored and replicated on various nodes in the network. Files stored in the network are identified by a location-independent globally unique identifier (GUID). To retrieve a file a search for the given GUID is propagated through the network. Once a node hosting a file with this identifier is found, the file is propagated back through the network to the original requester. Nodes through which the request and reply are

propagated also store a copy of the file. In this way popular files (i.e., those that are requested often) become more widely replicated than unpopular files. Freenet does not offer an explicit way to delete files, although unpopular files (i.e., those that are rarely requested) will eventually disappear from the network.

The Globe Distribution network (GDN) [2, 1] is a network for distribution of freely distributable software, developed by the Vrije Universiteit, Amsterdam and the NLNet Foundation. GDN is based on Globe distributed shared objects. Each object represents a software package and stores all the files contained in that package. A GDN object provides clients with an interface which allows files to be added, listed and removed. GDN also provides mechanisms to ensure that abusers of the GDN network (for example, those publishing non-freely distributable content) can be tracked and denied access to the network.

The Cooperative File System (CFS) [4] is a distributed file system based on the Chord distributed hash lookup technology developed at MIT. In CFS files are divided into blocks and blocks are distributed (and replicated) among available peers. The blocks are identified by hash values and these hash values are indexed in internal file system tables. retrieving a particular block requires using the Chord lookup technology to find a node that stores the particular block and retrieving it from that node. CFS is accessed as a regular file system using operating system system calls. As such, regular tools such as *cp*, *ls*, etc. can be used to access files in CFS.

# References


[1] A. Bakker, I. Kuz, M. van Steen, A. Tanenbaum, and P. Verkaik. "Global Distribution of Free Software (and other things)." In *Proc. SANE 2002*, May 2002. NLUUG.

[2] A. Bakker. *An Object-Based Software Distribution Network*. PhD thesis, Vrije Universiteit Amsterdam, Dec. 2002.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. "Freenet: A Distributed Anonymous Information Storage Service." In H. Federrath, (ed.), *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes on Computer Science*, pp. 46–66. Springer-Verlag, Berlin, 2001.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. "Wide-area Cooperative Storage with CFS." In *Proc. 18th Symposium on Operating System Principles*, Oct. 2001. ACM.

[5] G. Kan. "Gnutella." In A. Oram, (ed.), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, pp. 94–122. O'Reilly & Associates, Sebastopol, CA., 2001.

[6] B. Wiley. "Interoperability Through Gateways." In A. Oram, (ed.), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, pp. 381–397. O'Reilly & Associates, Sebastopol, CA., 2001.