GridDL: An HTTP Bandwidth Sharing Framework

Wolfgang Richter
University of Virginia
wor6c@virginia.edu

ABSTRACT

Peer-to-peer (P2P) applications have become a mainstream technology for content distribution. Yet widely used P2P applications, such as BitTorrent and Gnutella, suffer from flaws that are currently open topics of research—from the problem of freeriders to discrimination against peers with asymmetric Internet connections.

This paper presents the design of GridDL—a software framework for studying the new P2P paradigm that focuses not on exchanging *content*, but on exchanging *bandwidth*. GridDL provides researchers a tool to implement any algorithm for bandwidth sharing or bandwidth trading, while imposing as few limitations as possible on the researchers. GridDL creates a P2P network that executes a given algorithm for bandwidth sharing. GridDL also provides the first P2P bandwidth sharing application that does *not* require a pre-existing P2P network or infrastructure other than the individual peers.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*;
C.2.0 [**Computer Systems Organization**]: Computer-Communication Networks—*Data communications*

General Terms

Design, Experimentation, Measurement, Performance

Keywords

peer-to-peer, p2p, bandwidth, networking, protocol

1. INTRODUCTION

Bandwidth providers—Internet Service Providers, Universities, coffee shops—often impose artificial limits on the maximum amount of bandwidth any individual user may obtain to ensure a minimum quality of service for all of their users. Bandwidth limits have benefits; they prevent any single user from using all of the bandwidth, and they ensure a minimum amount of bandwidth for every user. However, bandwidth limits waste a network's physical capacity whenever the users do not use all of their guaranteed bandwidth.

Users, up until a couple of years ago, did not have a realistic option for circumventing artificial bandwidth limits. A new form of peer-to-peer (P2P) networking based on social bandwidth exchange [11] provides a solution. New research introduces the P2P paradigm of *bandwidth* exchange as opposed to *content* exchange. This P2P paradigm, if applied properly, promises to obtain full network capacity for all users with or without artificial limits. The ideal tool would provide users with a means of controlling their bandwidth and how it is allocated amongst them.

GridDL hopes to either become that tool, or motivate research that leads to such a tool. The incentive in a GridDL P2P network comes from *social* pressure—the promise of one peer to another that bandwidth will be repaid at some future date because the two peer owners know each other and, presumably, live close to one another. Social aspects of P2P networking have been explored before by the TRIBLER [11] P2P network, and in P2P protocols such as 2Fast [7], Amortized Tit-for-Tat [6], Give-to-Get [9], and Peer-Assisted Content Distribution with Prices [2].

Pouwelse et al. [11] identified five main research challenges for P2P networks: *decentralization* of functionality across the peers, *availability* guarantees, enforcement of *integrity* and *trust* between peers, proper *incentives*, and *network transparency* across NATs and firewalls. GridDL does not attempt to solve all of these issues. Instead, GridDL, and consequently this paper, focuses on *decentralization* and *network transparency* in support of P2P research. Implementations built upon GridDL are free to focus on any of the issues.

Of the current P2P networks, GridDL most resembles Firecoral [12]. Both GridDL and Firecoral use HTTP layer traffic to access web objects. Firecoral focuses on creating a full content distribution network that considers security, privacy, and usability [12]. Firecoral allows researchers to define any algorithm for peer-selection and chunk-scheduling, but it also requires extra infrastructure in the form of a trusted *signing service* [12] in order to function. GridDL has no such limitation and is a more general tool for research work.

This paper is organized as follows. Section 2 provides an overview of the design of the GridDL framework, and describes in detail the algorithm abstraction. Section 3 de-

U-NET'09, December 1, 2009, Rome, Italy.

scribes two algorithms for bandwidth sharing, one of which was chosen as the default algorithm included in GridDL. We conclude with performance observations in Section 4 gathered from both regular Internet traffic and contrived scenarios.

2. FRAMEWORK ARCHITECTURE

GridDL provides researchers with a minimalistic framework for implementing P2P networks using bandwidth sharing and trading algorithms. The framework includes a default algorithm that serves as an example and is discussed in §3. The GridDL framework has five main components: an algorithm library, a file system library, a P2P library, a UI library, and an HTTP library. Researchers only need to examine the algorithm library which provides them with an algorithm interface. GridDL uses implementations of the algorithm interface as the engine driving P2P interactions.

The HTTP library and the file system library provide abstractions to the rest of the framework for handling file I/O and HTTP I/O operations. The UI library contains interfaces for both console input and graphical user interfaces. The framework has been specifically designed to easily handle both types of user interfaces, and the UI library provides simple operations for using both types. The rest of this section is devoted to the Algorithm library, the P2P library, and the configurable parameters defined in the UI library.

2.1 Algorithm Library

The framework uses an *engine* to perform all operations related to classical P2P networks. The engine wraps around an algorithm interface which defines all of the operations of the engine. Implementing a new P2P algorithm is equivalent to implementing the algorithm interface and configuring the engine to use the new implementation. All algorithm implementations have access to a *peer tracker* data structure which maintains a set of all known peers and provides operations to the algorithm to get lists of peers or pick peers at random.

The algorithm library uses two abstractions defined within the file system library: *chunk* and *file*. Chunks are associated with a file, and they represent an arbitrary portion of their associated file (see §2.3). Files are associated with at least one chunk, and may be associated with many chunks. A file is considered downloaded once all of its associated chunks have been downloaded and has operations allowing the algorithm to obtain a list of chunks associated with it. The algorithm implementation is responsible for assigning chunks to known peers and is notified whenever a chunk finishes downloading.

The default algorithm implementations, discussed in §3, keep track of a *current set* of remote peers that they share bandwidth with and the local peer running GridDL. The current set could be equivalent to the set of all known peers, depending on the implementation of the algorithm interface.

The interface defines several abstract functions:

1. `download( )` - responsible for downloading a URL-accessible object
2. `setupPeerSet( )` - responsible for initializing the set of peers[1]
3. `retirePeer( )` - notifies the algorithm that a peer is no longer accessible
4. `newPeer( )` - notifies the algorithm that a new peer has been discovered
5. `finishChunk( )` - notifies the algorithm that a chunk has finished being downloaded
6. `downloadChunk( )` - notifies the algorithm that a remote peer requests the downloading of a chunk
7. `stopChunk( )` - notifies the algorithm that a remote peer no longer wants a chunk

Of course, algorithm implementors are free to create their own functions to support the required functionality of the algorithm interface. The main parts of the algorithm are invoked with `download( )` and `downloadChunk( )` which are the two operations that initiate downloads. Downloads initiated with `download( )` come from the local peer's user. Downloads initiated with `downloadChunk( )` come from remote peers and may be rejected through the protocol message REF, which is documented in Table 1 along with all possible P2P messages in the GridDL protocol.

2.2 P2P Library

The P2P library provides an implementation of a P2P network which operates automatically and provides the algorithm implementor with a black box surrounding all of the tasks required to create and maintain a P2P network. The P2P library provides two main services, *peer discovery* and *peer communication*, and two main abstractions, *the peer* and *the superpeer*. The core of the P2P library implements a symmetric, asynchronous, minimal protocol that peers use to communicate. All messages in the protocol are defined by their type and the data, or payload, that they carry. Thus, messages may be represented by the tuple:

$$\langle type,\ payload \rangle$$

All of the messages use TCP as the transport layer protocol, and IPv4[2] as the network layer protocol. UDP was considered as a potential method of bypassing firewalls, but UDP was deemed unnecessary with the introduction of a superpeer. Peers can tunnel through a superpeer which enables them to traverse both firewalls and NATs. Peers do not maintain persistent connections which distinguishes GridDL from popular P2P networks such as BitTorrent [3] and Gnutella [14]. The extra RTTs introduced by this method for TCP connection establishment and shutdown are considered negligible which allows the P2P library more freedom to decide when it wants to communicate with other peers. Short network outages may go unnoticed by peers because there are no persistent connections to interrupt.

2.2.1 Peer Discovery

When a peer first begins executing, it initiates the peer discovery phase contained in the P2P library. This phase handles discovering peers on the local subnet[3] and, if configured by the user, discovering peers within other subnets through an intermediary superpeer.

[1]Used by the default algorithms. Implementors do not *have* to define this.

[2]Adding support for IPv6 will be trivial

[3]Defined as the 254 potential network hosts obtained with a network mask of 255.255.255.0.

Table 1: P2P Protocol

Message Type	Description
NOP	Log the message data
HEL	Peer discovery initiation
SHU	Peer shutdown message
DOW	Request download indicated
STO	Stop download indicated
COM	Peer completed download
ERR	Peer error message
REF	Peer refuses download request
ACK	Acknowledgement to HEL

The first method of peer discovery uses the HEL and ACK messages as shown in Table 1. The peer performing discovery attempts to open a TCP socket to each IP on its subnet. If a socket is opened, the peer immediately sends an HEL message. Since the communication is asynchronous and persistent connections are not maintained, the P2P library considers a peer valid only upon receipt of an ACK message which must be initiated by the peer receiving an HEL and sent over a new TCP socket.

The second method of peer discovery uses an intermediary: the superpeer. Superpeers are responsible for maintaining a table of all peers that have contacted them through a TCP socket. When a new peer contacts a superpeer, the superpeer saves the new peer's IP[4] in its table and sends the connecting peer a list of known peers. After sending the list of known peers, the connection is immediately closed.

2.3 Configurable Parameters

The framework has two configurable parameters affecting an algorithm's implementation: *chunk size*, and *maximum connections*. Algorithm implementors are given control over how many peers to involve when sharing bandwidth. Chunk size defines the size of a chunk which is used by the file abstraction within the file system library to determine how to split a file into chunks. The maximum number of connections limits the number of outbound connections the algorithm may make to web servers—no parameter controls the number of inter-peer connections.

Increasing the maximum number of connections allows the framework to download more chunks in parallel, which can increase the rate at which a file downloads. Increasing chunk size reduces the amount of chunks, which reduces the amount of work that can be parallelized amongst the known peers. For example, if there are 10 peers and 10 chunks, then each peer may download 1 chunk in parallel, but if there are 10 peers and 1 chunk, then only one peer may download the file. Increasing the chunk size too much results in no distribution of the download. Of course, if chunk size is decreased to a very small value, such as 1 byte, than the overhead of TCP connections per chunk outweighs the benefits of distributing the download.

Algorithm implementors may ignore these parameters. The parameters were added to the framework to enable rapid testing for research. The parameters also enable users to have control over certain aspects of the framework. Modification of parameter values occurs through the UI library, which can take input from a console or a graphical user interface.

[4]Superpeers only save IPs that are not in the private use class of IPv4 address space.

3. ALGORITHM DESIGN

The algorithms were designed with the goals of being simple, and minimizing the download time of a file. Two separate candidate algorithms were developed during the implementation of the GridDL system. The first is a basic Round-Robin algorithm, and the second is a more advanced algorithm based on Estimated Average Bandwidth. The second algorithm assigns more weight to peers that appear to have higher bandwidth. This second algorithm is very similar to algorithms in conventional P2P networks, such as BitTorrent [3], and P2P networks designed around bandwidth sharing or trading such as [2, 7, 11].

The Estimated Average Bandwidth algorithm was chosen for inclusion in the final framework as both a default and an example because it outperforms the Round-Robin algorithm, potentially by orders of magnitude. The reasoning behind this choice is developed by the rest of this section. For analysis, we consider only one download in the system at a time, and we ignore the effect of peers opening multiple connections to a web server at once—this effect can be captured through the algorithm by measuring overall bandwidth coming from a peer.

3.1 Notation

In order to proceed with a rigorous understanding of these algorithms, we shall define several quantities:

T the time to download a file in seconds

P the number of peers in the current set, see §2.1

S the size of the file in bytes from an initiated download

μ the chunk size in bytes (a configurable parameter, see §2.3)

ψ_i the observed bandwidth of peer i in bytes per second

Ψ the observed total amount of bandwidth summed over all peers in bytes per second

Thus, total bandwidth is formally defined as:

$$\Psi = \sum_{i=0}^{P} \psi_i \tag{1}$$

Naturally, this is a simplification because we do not consider the bandwidth at the peer initiating the download that comes from remote peers in the current set. Using this simplification provides a good approximation and leads to simpler mathematical expressions.

3.2 Round-Robin

The first algorithm represents a naive approach to the problem of distributing the download of a web object across multiple peers, but it is simple and easy to implement. When a download is initiated through `download( )` (see §2.1) this algorithm assigns the chunks of the associated file to each peer in its current set in order until no more chunks are left to assign. This results in an equal number of requested chunk downloads to each peer in the current set.

The time to download a file using this algorithm is:

$$T = max\left\{\frac{S}{\mu \times P \times \psi_i}\right\} \text{ for every peer } i \qquad (2)$$

This algorithm is obviously inefficient, because it does not consider any individual peer's bandwidth ψ_i. As we can see, not considering a given peer's bandwidth can lead to slow download times—orders of magnitude slower than what they could be with optimal chunk assignment. In Equation 2, $\frac{S}{\mu \times P}$, may be considered a constant because none of the values in that expression vary with each peer i. Thus, the only value affecting time is the bandwidth variable ψ_i. As ψ_i decreases, the time T it takes to download a file of size S increases. Depending on the peers, values for ψ_i may vary by orders of magnitude which causes T to become large relative to peers with large ψ_i values.

3.3 Estimated Average Bandwidth

As hinted in the discussion of Round-Robin, an obvious improvement is to consider each peer's bandwidth ψ_i. The Estimated Average Bandwidth defines a new quantity:

$\frac{\psi_i}{\Psi}$ the fraction of overall bandwidth contributed by peer i, called *rank*

Rank is used to determine how many chunks to assign to each peer, which affects the `download( )` operation. In the default implementation rank is recalculated for all peers after a file finishes downloading (after all chunks for that file have been received).

Thus, the time to download a file has now become:

$$T = max\left\{\frac{S}{\mu \times P \times \psi_i} \times \left(\frac{\psi_i}{\Psi}\right)\right\} \text{ for every peer } i \qquad (3)$$

which simplifies to:

$$T = max\left\{\frac{S}{\mu \times P \times \Psi}\right\} \text{ for every peer } i \qquad (4)$$

This represents the ideal situation of full bandwidth utilization. As Equation 4 shows, the time it takes to download a file no longer depends on the peer with the least amount of bandwidth ψ_i; time now depends on the the total bandwidth Ψ available to all of the peers—the optimal solution for a P2P network.

4. EXPERIMENTAL ANALYSIS

Quantitative data was gathered through a custom test setup on a LAN subnet varying the number of nodes from 1-3 and separate tests involving three different files accessed over the Internet from different web servers. In addition, a real-world test was carried out using two Verizon FiOS connections within a neighborhood—although, the connections were on different subnets. All of the nodes in the LAN testing accessed a lighttpd web server [8] which limited each connection to 500 KiBps in download bandwidth. Each node was further configured to only open one connection at a time. This effectively limited each node's bandwidth to 500 KiBps.

As shown in Fig. 1, we observed an increase in average bandwidth per peer. This agrees with qualitative analysis of the Estimated Bandwidth Average algorithm in §3.3. The

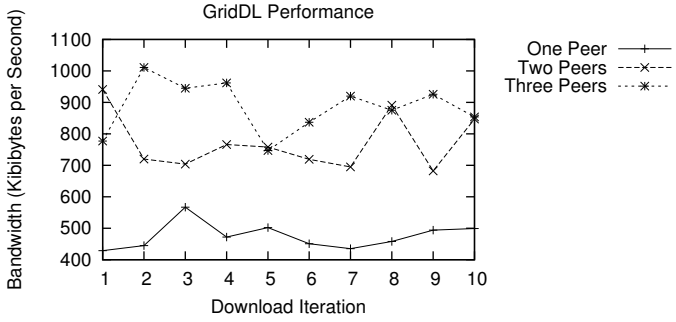

Figure 1: As the number of peers increased, average bandwidth per peer increased in the system.

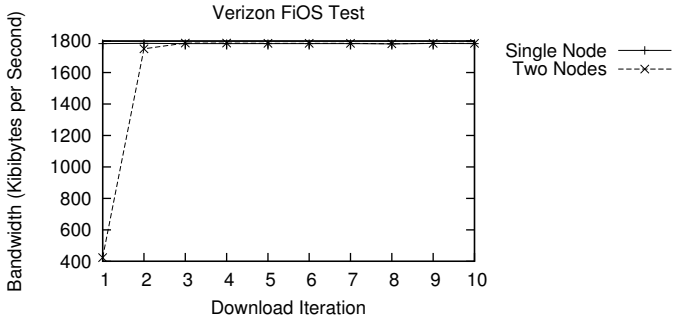

Figure 2: When using two FiOS connections, no significant increase in average bandwidth was observed.

test setup involved two peers using a shared wireless access point, and one peer using Ethernet to connect to the wireless access point. The file used for testing was [10].

Unless the wireless access point offers multiple clients high bandwidth at the same time, it is not a good choice for the control network. We conjecture that the wireless access point used in gathering data does not allow concurrent high bandwidth access, especially between wireless nodes. We further conjecture that this is the reason why we did not see as large an increase going from 2–3 nodes as we did from 1–2 nodes.

As shown in Fig. 2, we observed no significant increase in average bandwidth per peer as we added a peer to the GridDL network running on Verizon FiOS. This was expected, because the two peers are on different subnets and their routers are probably limiting their bandwidth as their communication traverses router boundaries. We expect that the average bandwidth per peer would increase if the peers are on the same subnet, but we did not have time to test this hypothesis on a real-world network.

Fig. 2 also shows the interesting effect of rank stabilization. The first couple of iterations involving two peers were significantly slower than the other iterations—essentially they were outliers. This shows how rank effects the average bandwidth of a peer as that peer observes the bandwidth contributions of other peers. However, the observed peer did not gain much from peer contributions. We conjecture that the observed peer did not gain from peer contributions because the network conditions were not ideal. The file used for testing was [5].

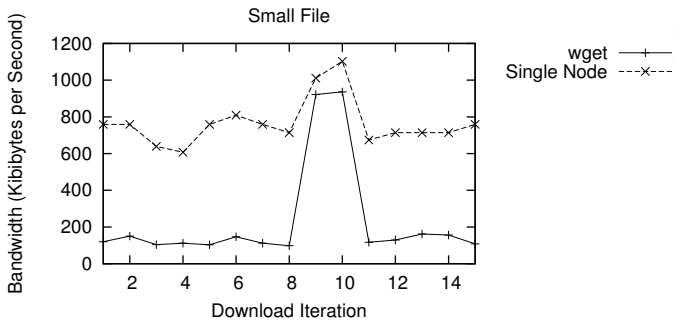

Figure 3: Single peer with multiple connections vs wget on a small file [10], 12 MB.

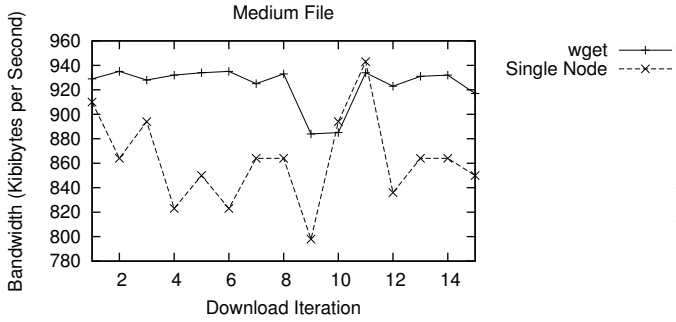

Figure 4: Single peer with multiple connections vs wget on a medium file [4], 50 MB.

The ideal setup for GridDL involves separate networks: an *Internet-connected network* per peer, and a *control network* shared by all peers. All of the GridDL peers would be connected to both networks—they would each have a route to the Internet and a high-speed *control network* through which all P2P traffic could travel without affecting Internet bandwidth. Of course, these two networks do not need to be separated, each peer could be connected to a high-speed network that limits their Internet bandwidth to a value much lower than their local network bandwidth.

Figs. 3, 4, and 5 show how the GridDL application running on a single peer compares to a singe-connection oriented application, wget, on the same peer when downloading files from the Internet. GridDL was limited to 15 connections, with a 1 megabyte chunk size, for all of the downloads, and the results were surprising, although they did agree with parallel TCP stream research [1]. The testing spanned several days. Testing began at 3 AM on a Wednesday and ran every 6 hours after that (3 AM, 9 AM, 3 PM, 9 PM). GridDL was scheduled to run immediately following the wget downloads, and each download ran sequentially after another download. We did not expect to find a large increase in average bandwidth for a single peer running GridDL. We thought large bandwidth benefits would not become significant until multiple peers joined the network. For the small file, GridDL outperformed wget in every case; in at least one instance GridDL was over 6 times faster. For the large and medium files, both GridDL and wget performed comparably and showed matching variations in bandwidth depending on the time of day.

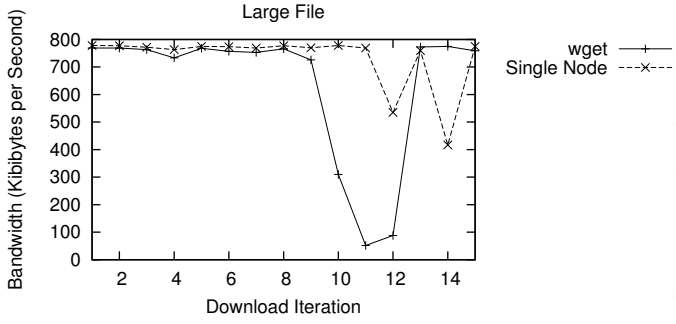

Figure 5: Single peer with multiple connections vs wget on a large file [13], 700 MB.

Quantitatively, our observations show that users of a bandwidth sharing P2P network can expect an average increase in bandwidth if certain criteria are met, such as having high bandwidth connections between the peers. In addition, we found that for at least one small file, GridDL consistently outperformed wget with only one peer. We conjecture that this is because GridDL was able to initiate a download of the entire file at once broken up into 12 chunks. With 12 TCP connections, GridDL was able to obtain more bandwidth than wget's single TCP connection. On the medium and larger files, GridDL did not have as much of an advantage because it had to continuously shutdown its TCP connections with the web server and re-establish them due to the limit of 15 concurrent web server connections.

5. CONCLUSION

This paper introduced a framework for P2P researchers studying bandwidth sharing and bandwidth trading. GridDL not only provides a framework for future research, but it also provides the first instance of bandwidth sharing occurring over HTTP connections. Previously, bandwidth trading or sharing required a BitTorrent client as in [11], or extra infrastructure through third-parties as in [12]. GridDL enables bandwidth sharing and, depending on the algorithm, bandwidth trading through normal HTTP traffic and it does not require any third-parties or that the content already be available in a pre-existing P2P network. The prototype framework implementation is almost feature-complete, and will be released on the web within the near future.

6. ACKNOWLEDGEMENTS

We would like to thank the Farrales family who generously donated time and bandwidth by allowing us the use of their Internet connection. Their aid helped confirm the viability of the GridDL P2P network on a real-world ISP network.

Additional thanks must go to Jason Lawrence, without whom this paper would not have come together in its present form, and Kamin Whitehouse, who's advice was very helpful.

7. REFERENCES

[1] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic. Parallel tcp sockets: simple model, throughput and validation. In *25th IEEE INFOCOM Conference*, 2006.

[2] C. Aperjis, M. J. Freedman, and R. Johari. Peer-assisted content distribution with prices. In *Proc. ACM SIGCOMM Conference on emerging Networking EXperiments and Technologies (CoNext '08)*, 2008.

[3] B. Cohen. Incentives build robustness in bittorrent. In *Proc. P2P-ECON*, June 2003.

[4] Damn Small Linux. `http://distro.ibiblio.org/pub/linux/-distributions/damnsmall/current/dsl-4.4.10-embedded.zip`.

[5] Fedora. `http://fedora.mirrors.tds.net/pub/fedora-/releases/11/Fedora/i386/iso/Fedora-11-i386-disc6.iso`.

[6] P. Garbacki, D. H. J. Epema, and M. Van Steen. An amortized tit-for-tat protocol for exchanging bandwidth instead of content in p2p networks. In *First International Conference on Self-Adaptive and Self-Organizing Systems, SASO*, 2007.

[7] P. Garbacki, A. Iosup, D. Epema, and M. van Steen. 2fast: collaborative downloads in p2p networks. In *6th IEEE International Conference on Peer-to-Peer Computing*, 2006.

[8] lighttpd. `http://www.lighttpd.net/`.

[9] J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, and H. Sips. Give-to-get: Free-riding-resilient video-on-demand in p2p systems. In *Proc. of SPIE - The International Society for Optical Engineering*, 2008.

[10] PHP. `http://us3.php.net/distributions/php-5.2.8.tar.gz`.

[11] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips. Tribler: a social-based peer-to-peer system. *Concurrency and Computation Practice & Experience*, 20(2), 2008.

[12] J. Terrace, H. Laidlaw, H. E. Liu, S. Stern, and M. J. Freedman. Bringing p2p to the web: Security and privacy in the firecoral network. In *Procs of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, 2009.

[13] Ubuntu. `http://mirrors.gigenet.com/ubuntu/intrepid/ubuntu-8.10-server-i386.iso`.

[14] Y. Wang, X. Yun, and Y. Li. Analyzing the characteristics of gnutella overlays. In *4th International Conference on Information Technology New Generations*, 2007.