P2P PROXY/CACHE

Ivan Klimek

Computer Networks Laboratory, Department of Computers and Informatics,
Technical University of Košice, Slovak Republic
Ivan.Klimek@cnl.tuke.sk

ABSTRACT

Internet as we know it nowadays is full of suboptimal behavior, widely used protocols are not optimized for factors not concerning the end users. This factors in end view affect the whole Internet, an example of such is the Peer-to-Peer (P2P) traffic problem. P2P traffic currently represents 50-90 percent [1] of the whole Internet traffic, with the expectation to grow by 400+ percent in the next 5 years [2]. Most Internet Service Providers (ISP) try to minimize this traffic using some form of restrictive counter measures, like traffic shaping, Fair Use Policies (FUP) or similar techniques. This approaches restrict the end user experience, limit the possibilities in which they are able to use their Internet connection just to save ISP resources or delay the need to invest into new infrastructure. This actions are artificially slowing down the growth of Internet which in fact as this study will show is not necessary.

KEYWORDS

P2P proxy cache, BitTorrent, Man-in-the-Middle attack, Network optimization, Content delivery

1. INTRODUCTION

P2P networks are based on the idea of decentralization, sharing of resources across large number of hosts/users. This decentralization enables the network to outperform all other content delivery technologies. In all of the parameters like total download amounts, speed, availability, scalability and cost - P2P superiority is unmatched. There are many P2P implementations currently available, however the most popular is the BitTorrent protocol with 60-90 percent of total P2P traffic [1]. BitTorrent is a very flexible protocol that can be used to deliver any kind of content. For example it is ideal for distributing Video-on-demand (VOD) and/or other high bandwidth applications. Because of the mentioned facts this study will focus on BitTorrent. To be able to fully exploit the possibilities of P2P networks we must first solve the negative side effects of decentralization. How can it be that the total P2P traffic is by more than 75 percent redundant [3]? This means the network communication is not effective and creates a lot of overhead.

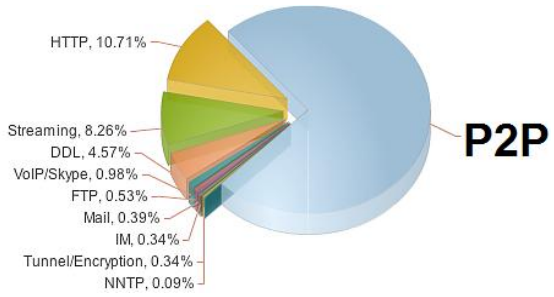

Figure 1: Protocol Type Distribution - Germany 2007, P2P represents 73.79 %

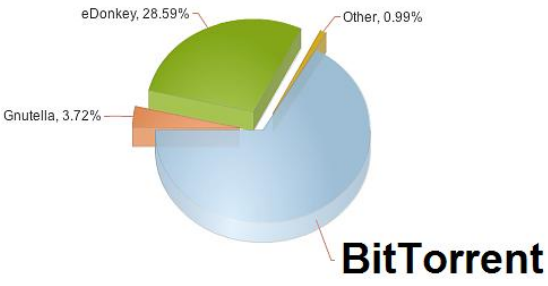

Figure 2: P2P Protocol Distribution by Volume - Germany 2007, BitTorrent represents 66.7 %

2. BITTORRENT ARCHITECTURE

The BitTorrent protocol became over the years of its existence pretty extensive that is why we won't describe its full architecture. We will focus only on the key element of this technology, the client-tracker communication. Tracker is a dedicated server that coordinates the clients; it does not hold any content itself. Clients download the metafile (.torrent) from the tracker resp. from other sources; open it via their BitTorrent client software which then reads the information contained in the metafile. This information can be divided into three categories: torrent information (date created, comment etc.) file information (name, length, hash) and tracker information (hostname, port). The client software then connects to the specified tracker and requests more information using a hash generated from the data contained in the metafile. This hash is called infohash and is used in the whole BitTorrent protocol to identify the torrent's content. Tracker replies with a list of other clients that are downloading the same content (leechers) or already finished downloading but are still uploading (seeders). The client software then starts the negotiation with other clients, but keeps updating the tracker in regular intervals. Except of this regular updates, the tracker is updated when the download is completed or paused too. The updates consist of information like: action (started, stopped, and completed), bytes received, bytes left, bytes uploaded etc.

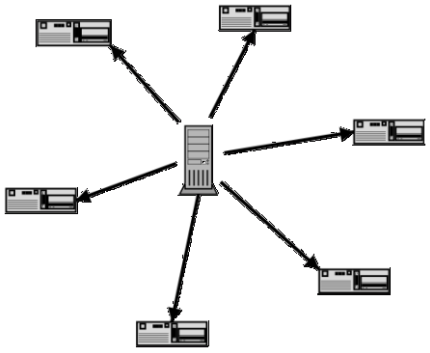

Figure 3: The Problem with Publishing: More users require more bandwidth

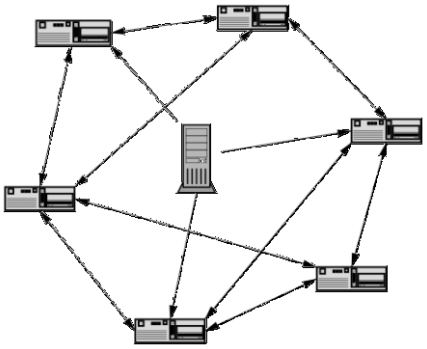

Figure 4: The BitTorrent solution: Users cooperate in the distribution

3. PEER-TO-PEER PROXY/CACHE

By evaluating the architecture of the BitTorrent protocol, we were able to detect an attack vector using which it is possible to perform an Man-in-the-Middle attack on the client-tracker communication. By intercepting the client requests and tracker responses, we were able to create a transparent BitTorrent proxy/cache server. This device gives us the full control over the user downloads in the network segment served by the proxy. It is possible to collect detailed statistical information on all downloads; select content to be cached and then transparently served to other users requesting the same content thus eliminating redundancy of data being downloaded.

Benefits of such approach:

- absolutely no uplink from clients in the given segment
- no redundant data outside of segment, massively reducing amounts of data downloaded
- if content is already cached, clients download the content at full speed of their Internet connection (link) from the very first byte to the end, this is in contrast to classic P2P behaviour where downloads start very slow and then by negotiating with more users gain speed

- if content is not yet cached and there is a high possibility more users will be interested in the same content the proxy actively supports the client's download thus greatly enhances the client download speeds while parallel caching the content

- because of the full control over client's downloads it is possible to optimize the network traffic

- specific content can be kept accessible longer than on classic P2P networks

Because the device is completely transparent it can be placed on different layers of the ISP's network, creating a hierarchical structure that maximizes the mentioned benefits even more. Using proxies hybrid P2P network architecture can be created, hybrid because the content is no longer completely decentralized, the data flows are no longer uncontrolled and by that the negative side-effects of decentralization are solved. The final effect of proxy-ing depends on fine tuning of the "all data - stored data" ratio. More disk space will enable more data being served from local cache. But when the content will be provided from the local cache, the clients won't need to upload. The content will be kept available by the proxy cache itself, so from the global perspective its availability will not be reduced. It is important to mention that because of the transparency of this solution neither the client software nor any other part of the currently used technology needs to be changed.

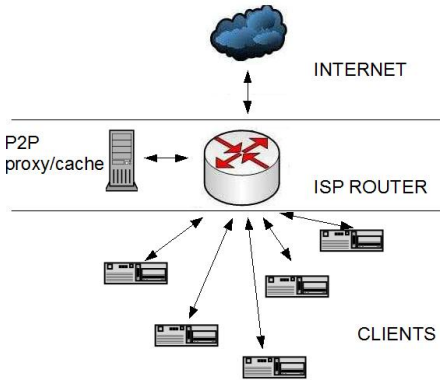

Figure 5: An example P2P proxy/cache topology

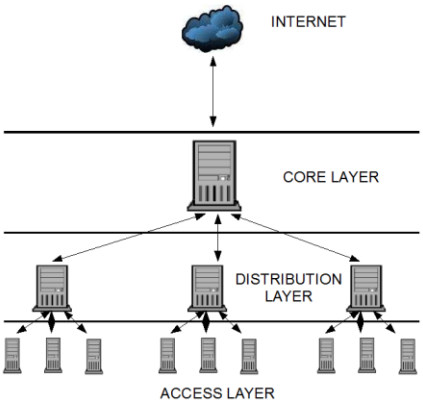

Figure 6: An example of a Hierarchical P2P proxy/cache topology

This technology can be used not only in standard wired or wireless ISP's networks but it also opens new possibilities for satellite Internet providers. Great majority of Earth's population has no access to broadband Internet. The latest generation of satellites is able to provide download speeds of up to 155Mbps with 6Mbps upload per user with relatively small dish size diameter of 45 centimetres [4]. The problem with this kind of Internet connection is the extremely slow latency generally about 500-900ms [5] (for comparison dial-up has latency around 150-200ms). The only way around this problem is to use on-orbit caching, thus reducing the latency for content provided from the cache to theoretical limit of 233ms (for the orbit height of 70,000 km). As shown with the use of extensive multi-protocol caching it would be possible to reduce latency to dial-up levels for almost all non-real-time traffic.

3.1. Intercepting Tracker HTTP/HTTPS protocol

Tracker requests are plain HTTP requests with a specific message format. For example some traffic filtering techniques use tracker responses to deny just the Peer-wire protocol (client-client communication). This is possible because they contain a clear text list of peers with their specific IP addresses and ports. As a reaction, some trackers started to encrypt this part of the

response. Our proxy recognizes and intercepts the requests, and then the actual man-in-the-middle (MiTM) attack begins. Most trackers still use HTTP or at least support it for compatibility reasons, however we are able to intercept HTTPS in most cases too. This is because many trackers use self-signed certificates which are vulnerable to MiTM. When this is not the case then MiTM on HTTPS can be still successful as majority of BitTorrent clients does not verify the server certificate. Another factor that guarantees that the proxy will be able to intercept the user request is that it is a de-facto standard to use always more than one tracker - multiplying the chance of interception. The only case in which this interception mechanism would not work would require that only one tracker is specified and that uses a CA signed certificate (or all specified trackers use HTTPS and have a CA signed certificate) simultaneously the BitTorrent client verifies the certificate and then actively refuses the connection because it detects a MiTM attack.

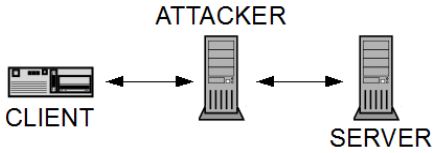

Figure 7: Man-in-The-Middle attack

3.2. The logic behind

The next phase after we intercepted the request is to decide if we already have the content cached. If yes, simply respond to the client instead of the original tracker providing the IP address of our proxy as the only peer who has the content. The port number provided in the response directly links on the application from which the requested content can be downloaded. If we do not have the content yet, we have to decide if it is "interesting" content i.e. it is viable to download it. This can be decided using the number of requests from the network served, for example if there will be more than x requests on this specific content then we cache it. Or we can use public search engines, look for how many people already downloaded it, if it is popular in the last x hours then it is highly probable there will be more than one request for that file from our segment and it is a good idea to cache it. While we cannot serve the requested content to the client, we forward the original tracker's respond to him. If we decided that it is an interesting content we will start to download it immediately, the client which originally requested the torrent is still downloading it and in regular intervals updating the tracker. In the next update he will get except of the "legal" peers the proxy in the list of peers too. The proxy will download the content faster than the client because of several factors:

- it is on a faster link
- it can accept incoming connections (firewalled client's speeds are lower because their ports are blocked)
- it updates its list of trackers using all possible trackers that can be found on search engines having the content to be downloaded registered

The content is being provided to the client parallel to downloading it, thus massively enhancing the client's download speed.

3.3. Finding content

We developed two methods of caching content without the need to dump or "record" any network communication. The default method is based on BitTorrent search engines. The client sends the infohash parameter in its requests, we use it to search for that specific torrent metafile

and then download it into the cache when needed. Using this method, it is possible to find most of the torrent metafiles, because the popular trackers are all well indexed. However, not everything can be found, for example if the content is served by some private tracker. When that happens, the backup method is used. This method is by far more complicated than the default one, but is able to reconstruct all the information needed to download the torrent content just from the client requests and some protocol hacking e.g. without the metafile itself. The problem is that the user starts the download based on information from the metafile which we do not have, so if we want to download the content we need to get the missing information somehow. The most important thing is to get the piece length, without this parameter it would be impossible to reconstruct the received data. Luckily, from information sent by other peers via the Peer-wire protocol it is easy to calculate the piece length parameter from the bitfield length. Combination of this two methods results in almost 100 percent probability of being able to download all requested torrents to the local cache.

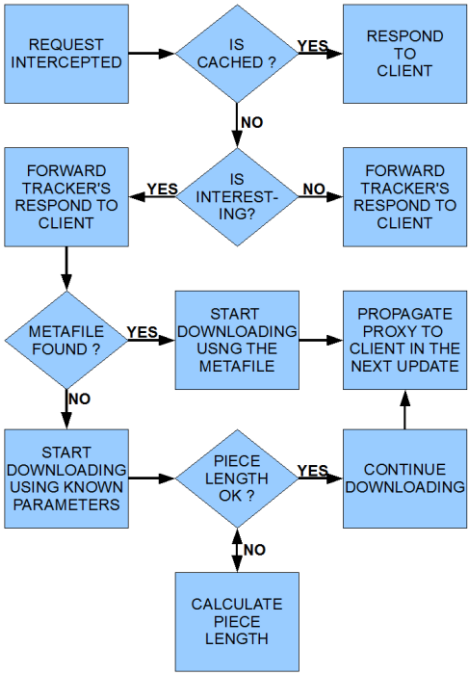

Figure 8: Simplified application logic

3.4. Full control

Because all BitTorrent downloads are being monitored by watching Tracker HTTP/HTTPS requests, it is possible to alter the traffic in ways that suit the current situation. For example because of the shared spectrum on wireless connections, P2P represents even a bigger problem. Few P2P downloads can burden the whole sector, making it very hard to keep the QoS for all users on acceptable levels. As all P2P traffic is being controlled, it is possible to add a layer of virtual time division multiplexing, which manages the transfers in a way that every time only one controlled data flow per segment exists. All clients recognize their sessions as active but in fact, they are getting data only few seconds per time period.

Benefits of such approach:

- P2P doesn't need to be restricted on wireless Internet connections
- P2P doesn't burden the sector excessively
- data speeds can be throttled dynamically based on network load

3.5. Software architecture

The Tracker HTTP requests are being intercepted by iptables [16] string matching rules. They are redirected to a virtual interface where tshark [14] with another much more strict string filter makes sure we intercept only the real Tracker requests not just requests containing the "right" key words. Tshark also parses the needed information for the MiTM attack. These arguments are then passed to Nemsis [15] which initializes the session hijack. From now on the proxy/cache is the Tracker for the client. The situation is basically the same for HTTPS, only the initial iptables matching cannot be made using string filter as it is not possible to see into HTTPS payload. Instead it is necessary to intercept requests based on "known trackers IP addresses". Modified Ctorrent [17] (a minimalistic console BitTorrent client software) is used for downloading the content. Every instance of Ctorrent is running on a separate port, this makes it easy to differentiate what content is available on what port. Ctorrent was modified to be able to download the content of a torrent without the metafile (this approach is needed only when the metafile cannot be found using BitTorrent search engines). It needs only the infohash, full length of the content and the tracker's address (one and more) parameters. All this information can be parsed from the clients request. Ctorrent then starts downloading the torrent with default piece length. It listens to Peer-wire protocol for the bitfields and using a special algorithm verifies if the default piece length is the right one. If not, then the download is restarted with the correct value.

The piece length calculation algorithm (C code representation):

```
size_t GetRealPieceLength(size_t bitfieldNBytes)

{

size_t pieceLength = argm_file_size / (bitfieldNBytes * 8);

return (log2(pieceLength)>int(log2(pieceLength))) ? 2^(int(log2(pieceLength))+1):
2^(int(log2(pieceLength)));

}
```

Where:

- bitfieldNBytes is the length of the bitfield
- argm_file_size is the full content size
- pieceLength is what we need to compute

According to the BitTorrent protocol specification: "A bitfield of the wrong length is considered an error. Clients should drop the connection if they receive bitfields that are not of the correct size, or if the bitfield has any of the spare bits set." [6] That is why we can be almost sure if we see few same bitfield lengths from different sources that this is the correct length. The Bitfield identifies which block has the client already downloaded. Piece length is usually power of 2 [6], that is why we simply look for the nearest bigger power of 2 to the division of full content length by number of bits in the bitfield. The problem is that the spare bits on end of the bitfield are set to zero, so its length does not need to be always the correct argument. That means the maximal error can be 7 bits, a very small probability exists our piece length calculation could provide a bad result. Anyway, this is solved later in Ctorrent code, where specific errors can be produced only by the piece length miscalculation; the piece length is then automatically divided by two. This should provide 100 percent correct setting of the piece length parameter.

Another problem with downloading torrents without the initial metafile is the process of checking for bad downloaded pieces as we do not have the SHA hashes to check the downloaded data with. This can be solved by another small modification of Ctorrent, as we are the only peer the client can download the content from (if it is already cached), when we provide a bad downloaded piece he will ask of it again and again. This behaviour can be easily recognized and the identified piece re-downloaded from the network and provided to the client. Another solution would be to add more sources (clients) into the Tracker reply except of us, so when the data cannot be downloaded from the proxy/cache it will be downloaded from a different source. As usually only a small percentage of blocks are bad the overhead created by adding more clients into the reply should be minimal.

3.6. HW requirements

The presented solution does not require special hardware; its requirements can be compared to normal file server for the given bandwidth. Although minor modifications consequent from the specifics of the protocol would greatly enhance its performance. The most heavily stressed component of a proxy is always the storage subsystem. Because the content will be mostly downloaded just once and then multiple times read the ideal approach would be to combine cheap high-capacity storage (using classic disks) with a extremely fast read buffer (using SSD technology). The downloading of content is a relatively slow process, thus it can be cached directly onto the high-capacity storage. From there it can be on-demand copied into the read buffer. The idea is to serve all requests from the read buffer. Because it has limited capacity only the currently most requested content is being held available on it.

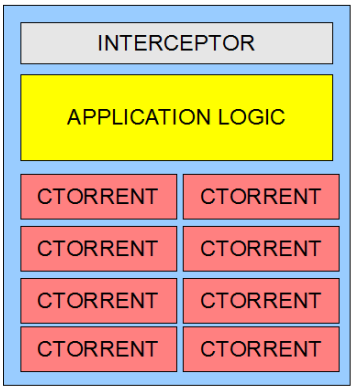

Figure 9: Simplified schematic representation of P2P proxy/cache software architecture

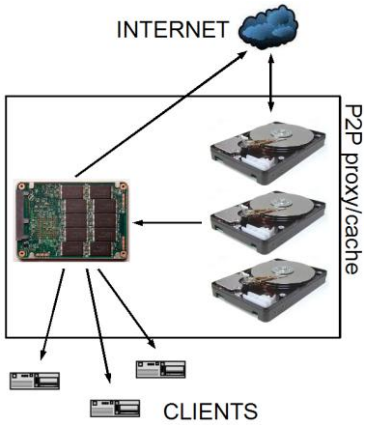

Figure 10: Schematic representation of the proposed storage subsystem, using SSDs for holding most frequently requested content

4. COMPARISON WITH SIMILAR TECHNOLOGIES

There are four main competitors, all with different approaches but all with the same goal - to solve the P2P traffic problem.

P4P - as described on its homepage: "P4P is a framework that can be used to enable Internet service providers (ISPs) and peer-to-peer (P2P) software distributors to work jointly and cooperatively." [7] This means a special dedicated server in the ISP infrastructure coordinates the P2P data flows, it also needs special client software.

pCache - academic open-source project, aims to create a P2P cache too. Different approach than ours, supports not only BitTorrent but Gnuttela too. It is a more complicated solution because it needs to monitor all P2P data, resulting in higher HW requirements. [8][9]

OverCache P2P Caching and Delivery Platform - short description from pCache authors: "Oversi's MSP platform realizes multi-service caching for P2P and other applications. An MSP

device actively participates in P2P networks. That is, MSP acts as a ultra-peer that only serve peers within the deployed ISP. We believe this approach negatively impacts fairness in many P2P networks, such as BitTorrent, which employ algorithms to eliminate free-rider problem. In fact, no peers in ISPs with Oversi's MSP deployed will ever upload anymore, because they expect to get the data free from the MSP platform. Once number of free-riders increases, the P2P network performance degrades, which in turns affects P2P users all over the world." [8] It is not a transparent solution.

PeerApp UltraBand Family - supports transparent caching of P2P traffic. Supported protocols are BitTorrent, Gnutella, eDonkey, and FastTrack. Uses Layer 7 protocol recognition (packet inspection) and according to that Layer 7 redirection to the application logic where it simultaneously to the client download acquires the file by dumping/cloning the transmitted data. Such solution is extremely HW demanding and we could say not optimal when comparing to our approach. [10]

	P4P	pCache	OverCache	PeerApp	P2P proxy/cache
Supports encryption	?	N	Y	N	Y
Transparent solution	N	Y	N	Y	Y
HW requirements	↓	↑	↑	↑	↓
Special SW needed?	Y	N	N	N	N

Figure 11: Comparison table

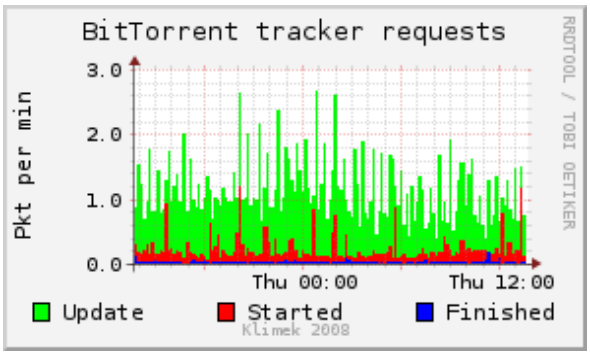

Figure 12: Monitoring of our university campus during 24 hours with approximately 3000 active users.

? - P4P uses its own SW

↑ - High HW requirements

↓ - Low HW requirements

The complexity of our approach compared to caching solutions that need to monitor all BitTorrent packets can be demonstrated on the ratio between the Tracker HTTP/HTTPS and the Peer wire protocols. On a example download with very small content size of 4.4 MB this ratio is 4 to 5988 i.e. 1 to 1497! In larger content size downloads this ratio would be even bigger although there would be some regular tracker updates send. In most situations, our P2P proxy/cache will need to process only one packet per download - the initialization "event=started" tracker request. The piece length checking should not be calculated in the complexity because it does not create any overhead as it is just a parameter check added into the actual content download process.

The rest of the proxy operation consists just of downloading and providing content, that means downloading a torrent and keeping seeding it until the cache capacity allows it i.e. it is not deleted to provide space for a currently more popular content. This is very similar to the Oversi's approach. Just with the exception that it is transparent for the clients, but acts as a peer for the rest of the Internet to keep the content globally available. This is for change similar to PeerApp technology except it does not need to use Layer 7 redirection. The transparent approach maximizes the caching effectiveness because it simply said "hacks" into the client download process and is not affected by the peer selection of the tracker or clients preference.

As of the character of the presented solution, it is impossible to provide statistical results, it can simply work or not work. The only statistical results that could be provided are associated with the popularity distribution, but because this topic has been researched by numerous previous studies [9][11][12][13] we won't focus on it.

5. CONCLUSIONS

The presented solution by exploiting a found attack vector in the currently most popular and perspective P2P protocol - BitTorrent, enables the ISPs to stop fighting P2P and instead cooperate with the community on creating a more effective hybrid P2P network model. The P2P proxy/cache is an extremely lightweight solution when comparing with other similar projects with at least the same or even better performance. It is important to mention that because of the transparent nature of this approach neither the client software nor any other part of the currently used technology needs to be changed. This technology has been already tested in several test scenarios reflecting real life situations, always with great results. We are completely confident about the usability of this device and are very close to installing it into our university campus network.

ACKNOWLEDGEMENTS

The author would like to thank: Associate Professor Frantisek Jakab PhD., Tomas Korenko, Bc. Marian Keltika and Martin Chalupka.

REFERENCES

[1] H. Schulze, K. Mochalski. Internet Study 2007. ipoque, November 2007.

[2] MultiMedia Intelligence, “P2P Traffic to Grow Almost 400% over the Next 5 Years, as Legitimate P2P Applications Become a Meaningful Segment”, 2008, [Online; accessed 23-November-2008], [Online], Available: www.multimediaintelligence.com

[3] PeerApp, “UltraBand Family overview”, 2007, [Online; accessed 23-November-2008], [Online], Available: http://www.peerapp.com/products-ultraband.aspx

[4] JAXA, “Overview of the KIZUNA (WINDS)”, 2008, [Online; accessed 23-November-2008], [Online], Available: http://www.jaxa.jp/countdown/f14/overview/kizuna_e.html

[5] Wikipedia, “Satellite Internet access”, 2008,[Online; accessed 23-November-2008], [Online], Available: http://en.wikipedia.org/wiki/Satellite_Internet_access

[6] TheoryOrg, “Bittorrent Protocol Specification v1.0”, 2008, [Online; accessed 23-November-2008], [Online], Available: http://wiki.theory.org/BitTorrentSpecification

[7] OpenP4P, “What is P4P”, 2008, [Online; accessed 23-November-2008], [Online], Available: http://www.openp4p.net/

[8] SFU, “Modeling and Caching of P2P Traffic”, 2008, [Online; accessed 23-November-2008], [Online], Available: http://nsl.cs.sfu.ca/wiki/index.php/Modeling_and_Caching_of_P2P_Traffic

[9] M.Hefeeda, C. Hsu, and K. Mokhtarian. Design and Evaluation of a Proxy Cache for Peer-to-Peer Traffic. School of Computing Science, Simon Fraser University, July 2008.

[10] PeerApp, “How P2P Caching works”, 2007, [Online; accessed 23-November-2008], [Online], Available: http://www.peerapp.com/products-ultraband-How-Caching-Works.aspx

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In Proc. of INFOCOM’99, pages 126–134, New York, NY, Mar. 1999.

[12] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In Proc. of ACM Symposium on Operating Systems Principles (SOSP’03), pages 314–329, Bolton Landing, NY, October 2003.

[13] M. Hefeeda and O. Saleh. Trafic modeling and proportional partial caching for peer-to-peer systems. IEEE/ACM Transactions on Networking, October 2007. Accepted to appear.

[14] http://www.wireshark.org

[15] http://nemesis.sourceforge.net

[16] http://www.netfilter.org/

[17] http://www.rahul.net/dholmes/ctorrent/