P2P proxy/cache

Bc. Ivan Klimek

Computer Networks Laboratory,
Department of Computers and Informatics,
Technical University of Košice, Letná 9, 041 20 Košice,
Slovak Republic.
Tel: +421-902-152873
E-mail: 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 ultimately affect the whole Internet, a great example of such is the Peer-to-Peer (P2P) traffic problem. P2P traffic currently represents 50-90 percent [1][2][7] of 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, thus artificially slowing down the growth of Internet which in fact as this study will show is not necessary.

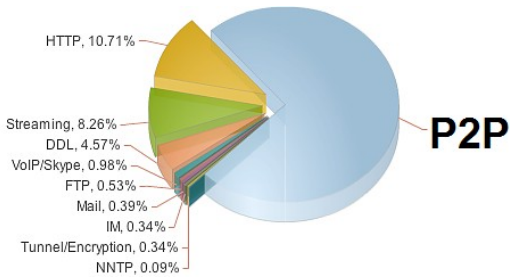

Figure 1: Protocol Type Distribution - Germany 2007, P2P represents 73.79 %

1 PERR-TO-PEER PROTOCOLS

P2P networks are based on the idea of decentralization, sharing of resources across large number of hosts/users. This decentralization enables the network to perform in ways that can not be compared to any other file/content providing technology. In all of the parameters as total download amounts, speeds, availability, scalability, costs - P2P superiority is unmatched. There are many P2P implementations currently available, the most used 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 the ideal way for distributing content like Video-on-demand (VOD) and/or other high bandwidth applications. Because of the mentioned we will focus on BitTorrent in this study. The main motivation is that the clear benefits and possibilities of this technology can not to be fully exploited without prior solving the negative effects of decentralization. The biggest issue is that the data flows are mostly not controlled, that means the whole network communication is not effective and creates a lot overhead. For example the total P2P traffic is by more than 75 percent redundant [3].

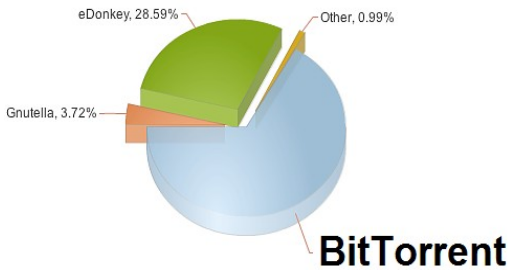

Figure 2: P2P Protocol Distribution by Volume - Germany 2007, BitTorrent represents 66.7 %

2 BITTORRENT ARCHITECTURE

The BitTorrent protocol became over the years of its existence pretty complicated, in this paper we wont try to describe the full architecture with all the extensions that currently exist. We will focus on the key element of this technology and that is 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

as 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 the torrent content using a hash generated from the informations 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 the download but are still uploading (seeders). The client software then starts the negotiation with other clients, but keeps updating the tracker in regular intervals. The tracker is updated when the download is completed too. The updates consist of informations like: action (started, stopped, completed), bytes received, bytes left, bytes uploaded etc.

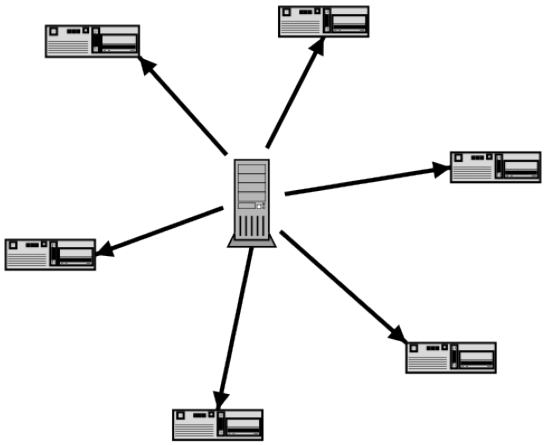

Figure 3: The Problem with Publishing: More customers require more bandwidth

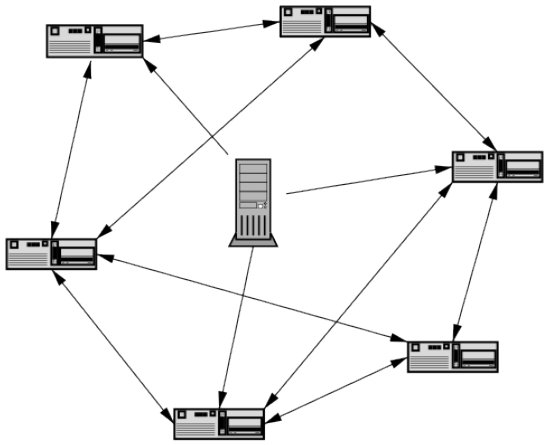

Figure 4: The BitTorrent solution: Users cooperate in the distribution

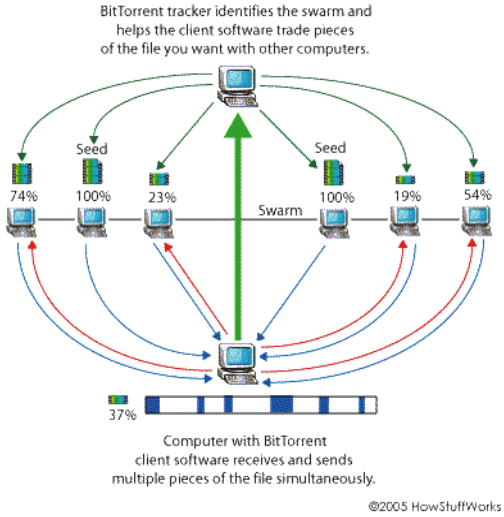

Figure 5: A graphical explanaition of the BitTorrent protocol architecture from HowStuffWorks.com

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 behind the proxy. It is possible to collect detailed statistical information on all downloads and based on this 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 in cache 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 behavior 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 download thus greatly enhances the client download speeds while parallel caching the content

- because of the full control over client 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 network creating a hierarchical structure that maximizes mentioned benefits even more. Using proxies a hybrid P2P network can be created, hybrid because the content is no longer completely decentralized, the data flows are no longer uncontrolled, the negative side-effects of P2P are solved. The final effect of proxying depends on fine tuning the "all data - stored data" ratio. More TB disk space will enable more data being served from local cache. Even that the clients wont upload any data (effectively freeing the last mile uplink) the P2P network from global perspective is not affected as the cache itself is keeping uploading the content. 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 has to be changed.

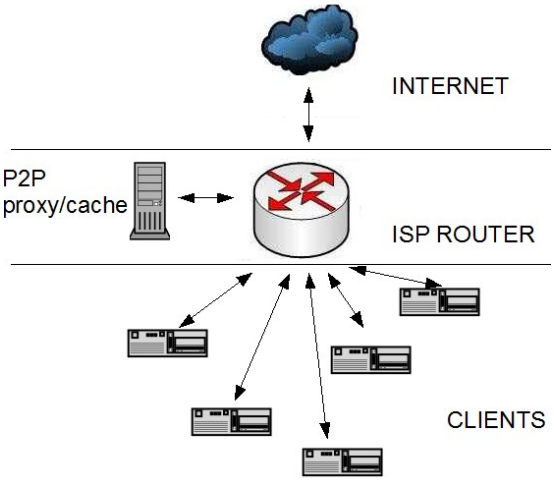

Figure 6: An example P2P proxy/cache topology

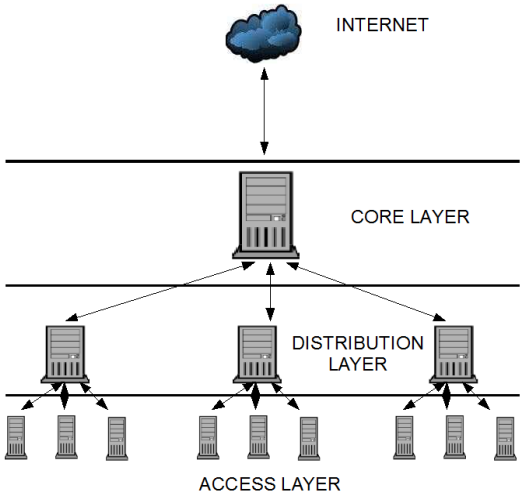

Figure 7: An example of a Hierarchical P2P proxy/cache toplogy

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 centimeters [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 the latency to dial-up levels for almost all non-realtime traffic.

3.1 Intercepting Tracker HTTP/HTTPS protocol

Tracker requests are plain HTTP requests with a specific message format, this could be called the completely weakest point of the whole technology. For example some traffic filtering techniques use tracker responses which contain a clear text list of peers with their specific IP addresses and ports to deny just the Peer wire protocol communication (inter-client communication). As a reaction, some trackers started to encrypt this part of the response. Our proxy attacks the requests, it recognizes them and intercepts them, then the actual man-in-the-middle (MiTM) attack begins. Most trackers still use HTTP or at least support

it for compatibility reasons[1], 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 because majority of BitTorrent clients does not verify the server certificate. Further, 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. As of the date of writing this paper the author is not aware of any BitTorrent client that would act as mentioned.

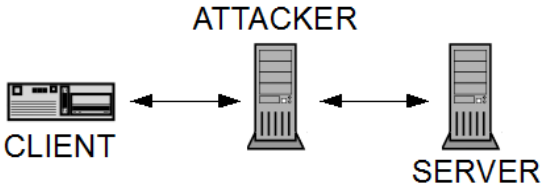

Figure 8: Man-in-The-Middle attack

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 as the original tracker providing the IP address of our proxy as the only peer who has the content, and the port number on which the content can be downloaded. If we do not have the content, 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 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 cant serve the requested content to the client, we forward the original tracker's respond to the 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 clients can still download but their speeds are lower because their ports are blocked so they can not accept incoming connections)
- 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 in 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. First, the default one 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. The problem is that the user starts the download based on information from the metafile, we do not have this metafile 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 informations 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.

1 It is a good idea to try if the same tracker is not working on HTTP in paralell.

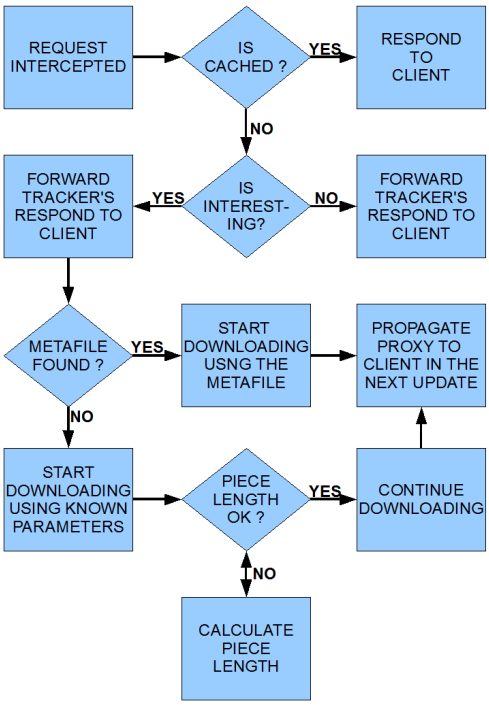

Figure 9: Simplified application logic

3.4 Full control

Because all BitTorrent downloads are being monitored by watching Tracker HTTP/HTTPS protocol, it is possible to alter the traffic in ways that suit the current situation/network technology/topology used. For example on wireless connections P2P represents a bigger problem then on wired because of the shared spectrum. 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, that manages the downloads in a way that every time only one controlled data flow per segment exists. All clients recognize their downloads as active but in fact, they are getting data only few seconds per time period[2].

Benefits of such approach:

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

2 The time frame for every client is calculated dynamically based on the maximal possible connection timeout and the number of clients in the sector.

3.5 Software architecture

The Tracker HTTP requests are being intercepted by iptables [16] string matching rules. The HTTPS requests can be intercepted only by iptables "known tracker IP addresses" matching because before the initialization of the MiTM attack it is not possible to see the actual request. The iptables rules change the destination to localhost where our proxy is listening for it. Communication with the clients is done using the popular netcat tool [14] (with stunnel for HTTPS [15]). After the request is intercepted, it is parsed and the application logic as described earlier is performed. For downloading the content modified ctorrent is used (minimalistic console BitTorrent client software). Every instance of ctorrent is running on a separate port, from that info we build a table telling on which port which content is being provided. Ctorrent was modified to be able to download the content of a torrent without the metafile (this approach is needed only when the metafile cant 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 of this informations 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 it 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[3].

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 him a bad downloaded piece he will ask for it again and again. This behavior can be easily recognized and the identified piece re-downloaded from the network and provided to the client.

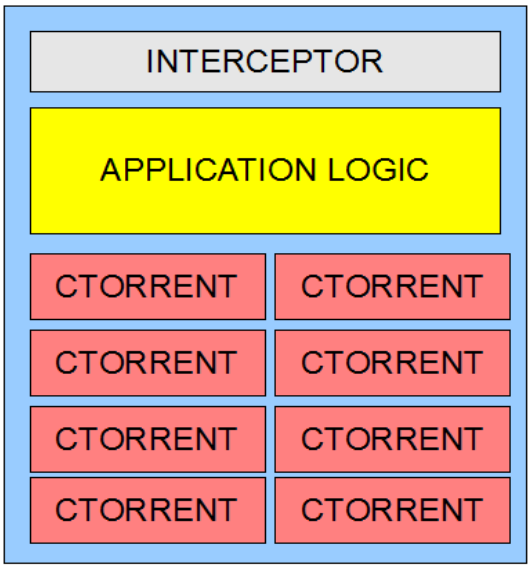

Figure 10: Simplified schematic representation of P2P proxy/cache software architecture

3.6 HW requirements

The presented solution does not need to be run on special HW, its requirements can be compared to normal file server HW for given bandwidth, although minor modifications consequent from the specifics of the protocol would greatly enhance its performance. The most heavily loaded component of a proxy is always the storage subsystem, but because the content will be mostly loaded just once into the cache and then multiple times read the ideal approach would be to combine cheap high-capacity storage (using classic disks) with extremely fast and extremely low access time read buffer (using SSD technology). The downloading of content is a slow process, everything is being cached onto the high-capacity storage, from that when needed copied onto the read buffer - SSD disk, all requests on that content are served from the buffer. The read buffer is dynamically cleaned so that only the currently most requested content is stored on it.

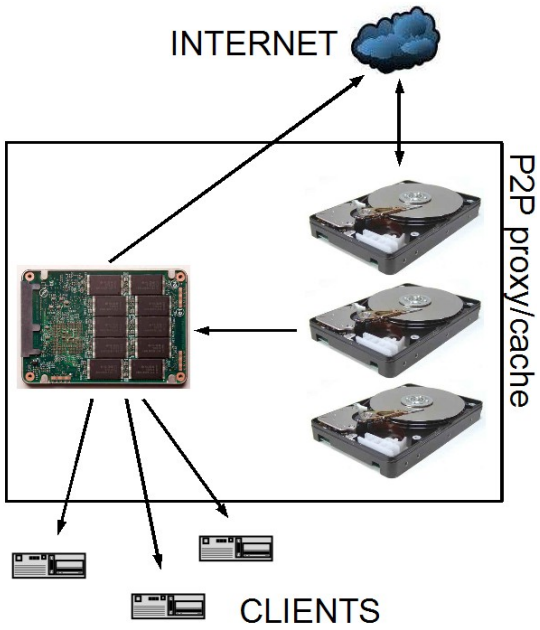

Figure 11: Schematic representation of the proposed storage subsystem, using SSDs for holding most frequently requested content

3 Except for cases when the piece length is not a power of 2. Which itself could be understood as an error.

4 COMPARISON WITH SIMILAR TECHNOLOGIES

There are four main concurrents, all have the same goal, to solve the P2P traffic problem but use different approaches.

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 describtion 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. Layer 7 protocol recognition is mostly being avoided on firewalls because of great performance impacts, thus we believe the PeerApp solution performance will suffer from it.[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 12: Comparison table

? - P4P uses its own SW
↓ - low HW requirements

↑ - high 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 check/process only one packet per download - the initialization “event=started” tracker request. In the worst case scenario, except of this one packet it will need to process the “event=completed” and the tracker update messages which still represents only few packets. 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.

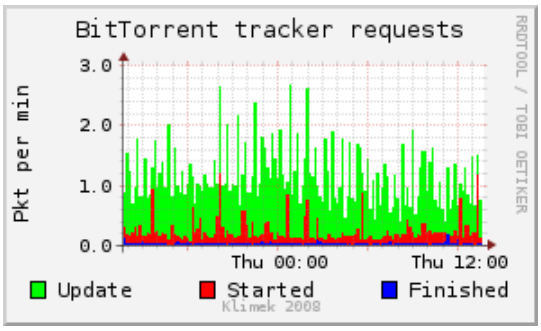

Figure 13: Monitoring of our university campus during 24 hours with approximately 3000 active users.

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 isn't 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 propagation/selection of the tracker or clients settings/preference. The achieved 100 percent success rate of interception[4] only demonstrates this.

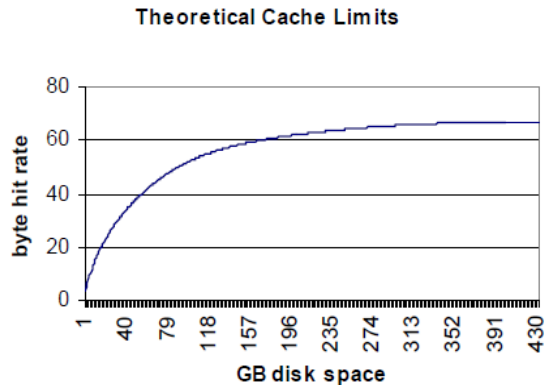

Figure 14: Popularity distrubitution diagram, a several TB disk field should perform up to 80 percent of byte hit rate [12]

4 In our tests, all connections (hundreds) that had to be intercepted, were successfuly automatically intercepted.

As of the character of the presented solution, it is impossible to provide universal true 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 wont focus on it.

5 CONCLUSION

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 a extremely lightweight and simple win-win solution when comparing with other similar approaches with at least the same or even better performance. 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.

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] N. Leibowitz, A. Bergman, R. Ben-Shaul, A. Shavit. Are file swapping networks cacheable? Characterizing P2P traffic, Expand Networks, Tel Aviv, 2002.
[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://netcat.sourceforge.net
[15] http://www.stunnel.org
[16] http://www.netfilter.org