LiteLoad: Content Unaware Routing for Localizing P2P protocols

Shay Horovitz and Danny Dolev
School of Computer Science and Engineering
Hebrew University of Jerusalem
Jerusalem, Israel
{horovitz,dolev}@cs.huji.ac.il

Abstract

In today's extensive worldwide Internet traffic, some 60% of network congestion is caused by Peer to Peer sessions. Consequently ISPs are facing many challenges like: paying for the added traffic requirement, poor customer satisfaction due to degraded broadband experience, purchasing costly backbone links and upstream bandwidth and having difficulty to effectively control P2P traffic with conventional devices.

Existing solutions such as caching and indexing of P2P content are controversial as their legality is uncertain due to copyright violation, and therefore hardly being installed by ISPs. In addition these solutions are not capable to handle existing encrypted protocols that are on the rise in popular P2P networks.

Other solutions that employ traffic shaping and blocking degrade the downloading throughput and cause end users to switch ISPs for a better service.

LiteLoad discerns patterns of user communications in Peer to Peer file sharing networks without identifying the content being requested or transferred and uses least-cost routing rules to push peer-to-peer transfers into confined network segments. This approach maintains the performance of file transfer as opposed to traffic shaping solutions and precludes internet provider involvement in caching, cataloguing or indexing of the shared content. Simulation results expresses the potential of the solution and a proof of concept of the key technology is demonstrated on popular protocols, including encrypted ones.

1. Introduction

In its early days, the Internet was used to transport fairly homogeneous and relatively light textual content between computers. Since then, the Internet has become an extremely diverse platform which is used to transfer a myriad of different types of content in a variety of formats. Today, bandwidth intensive content, such as media files, for example, traverses the Internet alongside relatively light content, such as text and simple graphics.

Peer to peer (P2P) technology is a major contributor to the dramatic rise in the amount of bandwidth intensive content being exchanged over the Internet. P2P services, such as eDonkey/eMule and BitTorrent are believed to be responsible for approximately 60% [10] of all Internet traffic. Internet Service Providers (ISPs) typically pay for external links (links with nodes located outside the specified portion of the IPS's network). Thus, P2P traffic, being associated with content intensive traffic, is a significant contributor to the operating costs of ISPs and has become a heavy financial burden on the shoulders of the ISPs. In addition, the quality of the service is degraded as heavy P2P users exploit the available bandwidth for external throughput.

If ISPs were provided with effective means for reducing the operating expenses associated with P2P traffic, their profitability may increase. Such solutions need to maintain a relatively high quality of service level due to the popularity and demand of P2P services. Therefore, simply blocking P2P traffic or employing a traffic shaping mechanism is not feasible. In an attempt to provide effective means for reducing the operating expenses associated with certain types of traffic, in particular - P2P traffic, several solutions have been suggested such as using cache servers and content-aware P2P routers, all aimed to localize P2P traffic inside the IPS's network. However, such solutions expose the ISP to the content of P2P communications and even involve storing (caching) data which is associated with (or refers to) the content of the P2P communications. Exposure to the content of the P2P communications may impose considerable liabilities on the ISP, for example, in lieu copyright infringement. In addition, recent versions of P2P protocols are encrypted - thus removing the benefit of cache servers and other routing solutions that try to inspect the content of the packets.

Thus, there is a need for a method and a system for content insensitive management of P2P communications arriving from or to a node connected to a specified portion of a network. It is further needed to provide a solution for managing communications arriving from or to a node connected to a specified portion of a network without being exposed to the content of the communications and without being required to store or cache any data which is associated with or including reference to the content of the communications. In addition, support of encrypted P2P protocols is required.

We present the design of LiteLoad, a system that provides a method for managing communications arriving from or to a node connected to a specified portion of a network or an ISP in particular. LiteLoad does not store any information about the content being transferred between P2P nodes. The concept was tested in a simulation that showed a fundamental advantage. In addition, we built a proof-of-concept prototype for some portions of the system and proved its ability to perform over popular P2P protocols, both encrypted and non encrypted.

LiteLoad's approach to the problem is to learn the patterns of communications of P2P protocols and accordingly apply rules that influence the behavior of such protocols for creating a localized network.

The remainder of this paper examines these issues both analytically and empirically. In Section 2 we discuss related work in this field. Section 3 elaborates on the problem and the different aspects of the tention between ISPs and P2P networks. Section 4 presents our approach and discusses how and in what scenarios this approach will benefit the ISP and its customers. Section 5 presents our experiments on popular protocols and the results of our simulation.

2. Related Work

In order to solve the problem of P2P bandwidth congestion on external links of ISPs, several solutions have been proposed. Aggarwal et al. [13] proposed employing an oracle service. P2P clients should supply the oracle with a list of possible P2P neighbors and the oracle ranks them according to certain criteria such as their proximity to the requesting client.

In [17] the authors propose a caching solution which leverages existing web cache proxies. Yet, the P2P client should be adapted to this service.

The above solutions are not able to deal with existing P2P networks and require the designer of the network to adapt its architecture to these solutions.

Solutions that were offered by CacheLogic [3], PeerApp [11] and Joltid [8] proposed to cache P2P traffic. This makes sense as in P2P it was shown that 20% of the files account for more than 80% of the downloads [16]. Yet, caching means that portions of copyrighted material are stored on the ISPs disks and this raises a legal dispute about copyright violation. In addition, due to the new trend of encrypted protocols with changing keys and the use of obfuscated packets, caching seems to be losing its grip in the market.

Gummadi et al [14] offered that an ISP will deploy a redirector at its boundary. The redirector would index the locations of objects (shared files) on peers within the ISP and route internal clients' requests to other internal peers whenever possible. Such a solution might expose the ISP to frequent demands from the music and film industries to block the delivery of copyrighted material. Yet, ISPs are not interested to prevent access to copyright infringements as their users might leave to a competing ISP, therefore - any solution that will make an ISP "aware" of copyrighted material is not acceptable. Another offer presented in the same paper related to current P2P systems that use supernodes. It was offered to employ a topological distance estimation techniques to make it possible to infuse supernodes with locality awareness. Still, this solution requires a change in the code of supernodes and will not work on existing networks or future networks that will not implement it.

Another routing-based solution was offered by Sandvine's Peer to Peer Element [12]. A device is installed by the ISP and it acts as a large scale supernode, serving internal nodes and trying to match queries with content that is being shared locally. This solution requires deep reverse engineering of each P2P protocol as it mimics the supernode. This requires the ISP to continuously update the device for changes in each protocol. In addition, since it acts as a supernode, it holds a list of shared files and the addresses of peers that share them, and by doing so, it makes the ISP "aware" of copyrighted infringement. Another weakness of Sandvine's solution is that it can't support encrypted protocols where it's not possible to reveal the requested query of the client.

3. The Problem of P2P congestion in ISPs

Many solutions were proposed to solve the problem of P2P bandwidth in ISPs yet there is evidence that this problem was not solved yet.

Though many ISPs still deny employing solutions for that problem, recently (Nov, 07) Canada based Bell Simpatico has confessed [5] to using "traffic management" on heavy users during peak hours. According to their administrator, Bell Simpatico's traffic shaping affects several applications and protocols including BitTorrent, Gnutella, Limewire, KaZaA, eDonkey, eMule and WinMX.

Only a month earlier, an independent testing [6] performed by Associated Press revealed that Comcast, the second largest ISP in the US interferes with P2P traffic going to and from its high-speed internet subscribers, by impersonating users' machines and sending fake disconnect signals.BitTorrent Inc confirmed these findings and noted that similar practices were already seen from several Canadian IPSs.

A recent survey from traffic-management company Ipoque [7] show that P2P traffic range between 50% up to 90% of all Internet traffic. File sharing applications like eMule/eDonkey and Bittorrent dominate with shares of up to 75% of all P2P traffic (depending on geographical zones).

For better understanding the size of the problem and learn the required properties for a feasible solution, we visited several ISPs. Following is a summary of the most important properties in the eyes of those who face the problem on a daily basis:

1. Must be oblivious of the content being shared and/or transferred between nodes;
2. Enforce both exposed and encrypted protocols;
3. No additional new protocol should be required;
4. Automatically support protocol updates as much as possible.

Several studies [14, 15] showed that bandwidth savings of the order of 60% are achievable by exploiting traffic locality in P2P. Therefore, using cache or a global index of shared files and employ redirection is reasonable. Yet, these solutions are controversial as the ISP would no longer be merely providing an infrastructure but also store and forward copyrighted content - that leads the ISP into copyright violation. Thus, many ISPs avoid using these solutions and prefer traffic-shaping solutions such as those offered by Allot [1], Cisco (PCube) [4] and Packeteer [9]. Traffic Shaping solutions can be recognized by end users as they experience poor performance compared to their friends that are connected to a different ISP. Azureus(BitTorrent client) for example, maintains a list [2] of ISPs that use traffic shaping and provides a set of simple steps how to avoid that. In addition, as we mentioned earlier in this document current methods can not handle encrypted protocols as they require either a substantial reverse engineering of the protocols, deep packet inspection filtering.

4. LiteLoad

The previously discussed problems motivate the need for a new approach that makes it possible to localize P2P protocols without being aware of the requested and transferred content. Therefore, it should neither employ caching of content nor maintain tables or lists where one can deduce that specific portions of a given content is being shared by specific user as it contradicts the interests of the ISP. In addition, it should also handle encrypted protocols. Existing caching solutions or any other solutions that perform reverse engineering of the protocol cannot handle encrypted protocols.

4.1 Behavioral Patterns

LiteLoad is a solution for managing communications arriving from or to a node connected to a specified portion of the network without examining the content being transferred. Instead, it looks for patterns of communication that match existing P2P networks' patterns.

For example, we examine the communications pattern of a browser that browses a simple HTML page that contains an image. If we monitor the network activity we will find that:

1. Client asks for index.html;
2. Server returns index.html;
3. Client asks for image1.gif;
4. Server returns image1.gif;

This pattern is very simple and it looks like one can hardly learn anything from it. Yet, if we monitor the same server and notice this sequence:

1. Client asks for index.html;
2. Client asks for index.html;
3. Server returns index.html;

we know that there was a problem in the process that made the client ask for the file twice. However, if there was a large time gap between the first two requests, we can consider it as normal. By employing similar techniques we can test P2P protocols and learn their behavior on different scenarios. Once we are familiar with a protocol behavior, we can trick it to perform various actions, depending on the protocol and topology. In this work we demonstrate these techniques and show how they can support us in localizing P2P protocols.

4.2 Structure

We first describe informally a typical simplified scenario of LiteLoad to localize a supernode based network (such as eDonkey/eMule and KaZaA): When a user runs the P2P client application, the application tries to connect to a supernode. We term the first message that is being sent to a potential supernode by the client as *session initiation message*. This message tells the supernode that the client requests to register to it and later on query it for desired content. LiteLoad intercepts all session initiation messages of a P2P protocol and checks whether the destination address (of the requested supernode that the client tries to connect to) is internal or external to the ISP's network. In case it's internal, LiteLoad lets the message reach it's original destination. In case it's external, LiteLoad alters the header of the message to a new destination of an internal supernode. Now assume that the client didn't find the file it looked for, then it will try to connect to a different supernode within a short time. LiteLoad stores the time stamps of the client's session initiation messages so it can recognize that the client failed to find its requested content in the previous connection to a supernode; therefore, for the current request, LiteLoad will let the new session initiation to proceed to its original destination (that may be external to the IPS's network). This type of behavior pushes the P2P clients of an ISP to try internal supernodes first. Since eventually internal supernodes will serve mainly internal clients, a substantial part of file transfers will be served by internal clients. Notice that the policy we chose to replace the address is content insensitive as we only modified the header of a session initiation message. Moreover, we do not intercept the actual content in the messages.

Other behavioral patterns may be relevant for the process. For example, a user that looks for 10 different files in a short period of time might appear to the system as if it received no search results. In most protocols this is not a problem since the search message is different than the session initiation message, thus there's still one session initiation message per all searches. However there might be future protocols where each search will initiate an initiation message to other supernodes thus recognizing the behavior of each protocol is crucial.

In addition, a user behavior might affect the decision modules of the system as in EMule some users manually switch supernodes. It's possible to recognize behavior per user and decide how to perform per each behavior pattern under different scenarios.

4.2.1 Address Replacement Module

We now refer to Figure 1. *P1* is a P2P client application that was asked to download a specific file *F*. LiteLoad is installed at ISP *X*'s premises and is represented at the center of ISP *X*'s cloud. The *Filter* resides inside the ISP's router and forwards certain P2P messages to LiteLoad. For example, in a supernode-based network, the *Filter* will forward only session initiation messages between clients and supernodes (which is the first message that a client sends to a potential supernode for registration). The *Filter* may be configured to determine that a certain message is a session initiation message by reading its header.

Message *D* represents a message that did not comply with the interception criteria implemented by the *Filter* and therefore it is allowed to pass through the *Filter* substantially uninterrupted to proceed to its original external destination address, which is associated in this case with node *P5*. Upon receiving a message from the *Filter*, the message or certain data relating to the message may be input to the *External Link Identifier* - which is adapted to identify whether messages' destination address is external to ISP *X* by reading the message's header. In case that the address is internal to ISP *X*, the *External Link Identifier* will allow the message to proceed to its original destination unchanged. However, if the *External Link Identifier* identifies a P2P session initiation message with a destination address that is external to ISP *X*, the message will be forwarded to the *Address Replacement Module* as in the case of messages *A*,*B* and *C* in our example. In Figure 1, message *A* includes the address of external node *P6*, message *B* includes the address of external node *P4* and message *C* includes the address of external node *P5*.

The address replacement module may be adapted to determine whether an address or addresses included in an identified P2P session initiation message should be replaced in accordance with a content insensitive replacement policy. Some examples of various replacement policies which may be implemented by the address replacement module where provided above. As already mentioned above, the address replacement module may use the replacement policy to determine if and when an address included in an identified P2P session initiation message should be replaced. The address replacement module and the replacement policy utilized by it, may relate to information found in the header of an identified P2P session initiation message, and the address replacement module may not access the content of the message. Thus, the address replacement module may not become exposed to the actual content or payload of an identified P2P session initiation message.

4.2.2 Address Pools

LiteLoad includes at least one pool of replacement addresses. In the sample presented in Figure 1, two pools are included. The first pool is a pool of internal addresses, and the second is a pool of external addresses. The inclusion of an address in either of the pools may be insensitive to the content of the address as it is being collected from previous headers only. LiteLoad may include a pools management module. The pools management module may be adapted to create and manage each of the internal and external addresses pools. Each of these pools may be associated with a different preferred address criterion. The pool management module may be configured to determine whether a candidate address should be included in the internal or the external pool in accordance with the preferred address criterion with which each of the pools are associated. The certain portion of the network ISP *Y* to which the criteria associated with external addresses pool relates may be, for example, a network of another ISP (or other organization) with which external links are relatively cheap.

Notice the LiteLoad also includes an internal candidate

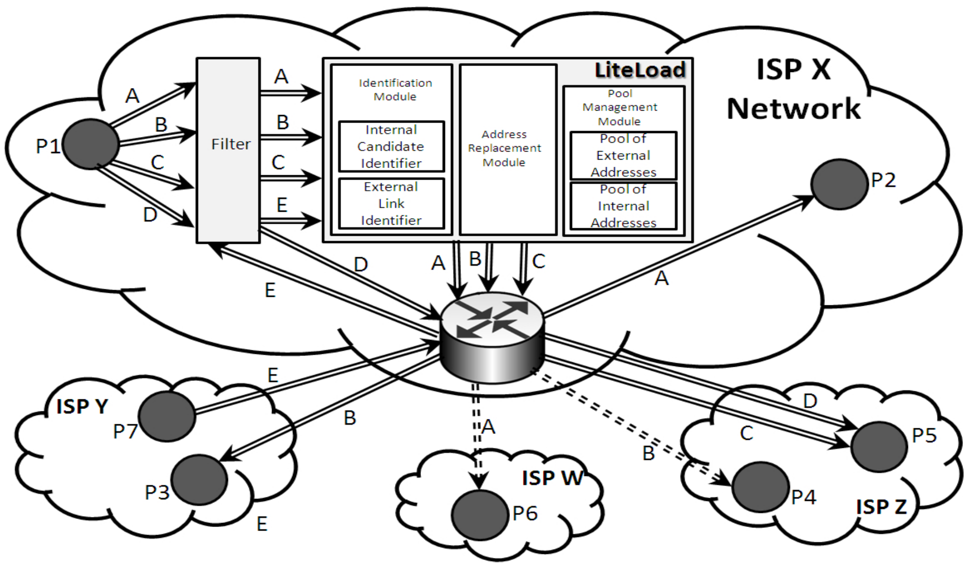


Figure 1. LiteLoad Architecture

identifier. The internal candidate identifier may be configured to receive from the filter messages which have been intercepted by the filter or data with respect to such messages. The internal candidate identifier may be configured to read the header of a message intercepted by the filter , or corresponding data, to determine whether the header includes an internal address. If an internal address is found in the intercepted message, the internal candidate identifier may forward the data with respect to the candidate address to the pools' management module. The pools management module utilizes further criteria to determine whether the candidate address should be added to the pool. For example, the pools' management module may check whether the candidate address is already included in the pool.

The external candidate identifier is configured to read the header of a message intercepted by the filter , to determine whether the header includes an external address that is within a certain portion of the network. If an address from within that certain portion of the network ISP *Y* is found, the internal candidate identifier may forward the data with respect to the candidate address to the pools' management module, which utilizes further criteria for determining whether the candidate address should be added to the pool prior to the inclusion thereof in the external pool.

As in the example of supernode based networks,in accordance with the replacement policies implemented by the address replacement module, an external address included in an identified message (P2P session initiation message) should be replaced, by default, by an address from the pool of internal address. Furthermore, an external address included in an identified message should be replaced with an address from the pool of external addresses, if the identified message follows a previously identified message from the same source, in this case from node *P1*, and the interval between the identified message and the previously identified message is less than a first threshold. An external address included in an identified message should not be replaced and the identified message should be allowed to proceed to its original destination without being modified, if the identified message follows at least two previously identified messages from the same source, and the interval between each identified message and it predecessor is less than a second threshold.

4.2.3 Message Flow Scenarios

Message A, B and C, illustrate the application of the above exemplary replacement policies. P2P session initiation message A, which is the first of the three to arrive at LiteLoad, is determined to include an address that is external to the ISP *X*'s network, for example, the address of node *P6*. Upon receiving data with respect to message A, the address replacement module may determine with which of the replacement policies message A complies. Since at the time of identifying message A the address replacement module is not aware of any preceding identified messages from the same source (node *P1*), the default replacement policy is implemented by the address replacement module, and a replacement address is selected from the pool of internal addresses for replacing the external address included in message A. Subsequently, message A is rerouted in accordance with the internal replacement address, in this case to internal node *P2*.

P2P session initiation message B, which arrives at LiteLoad shortly after message A was identified, is determined to include an address that is external to the ISP *X*'s network, for example, the address of node *P4*. Upon receiving data with respect to message B, the address replacement module may determine with which of the replacement policies message B complies. Since message B follows a previously identified message from the same source, in this case message A, and the interval between the identified message E and the previously identified message A is less than the first threshold, the address replacement module determines that the external address included in message B, in this case the address of external node *P4*, is to be replaced with an external address selected from the pool of external addresses, and in this case with the address of external node *P3*. Subsequently, message B is rerouted in accordance with the external replacement address, in this case to external node *P3*.

P2P session initiation message C, which arrives at LiteLoad shortly after message B was identified, is determined to include an address that is external to the ISP *X*'s network, for example, the address of node *P5*. Upon receiving data with respect to message C, the address replacement module may determine with which of the replacement policies message C complies. Since message C follows two previously identified messages from the same source, in this case messages A and B, and the interval between each identified message and it predecessor (in this case between C and B, and between B and A) is less than a second threshold, the address replacement module determines that the identified message C should be allowed to proceed to its original destination, in this case to external node *P5*, without being modified.

4.2.4 Access Hash Table

Figure 2 is a graphical illustration of a hash table that is used by the address replacement module to store data with respect to identified messages. The hash table may be a fixed size table that includes a fixed number of hash value entries (the table in the example contains 500 entries). Each value in the table may represent one or more nodes from which an identified message was received. Each hash value may correspond to the result of a hash function, when applied, for example, by the address replacement module, to the IP address of the nodes from which the identified message was received. For each value entry, a timestamp may be stored. The timestamp is stored in connection with a certain hash value that may correspond to the time when the most recent message from a node whose address corresponds to that hash value was identified. In addition, the hash table may further include for each hash value entry a counter value. The counter value parameter corresponds to the number of messages received from a node whose address corresponds to the hash value, which have been identified within a predetermined period from the identification of a previously identified message from a node whose address corresponds to the same hash value. Thus, for example, whenever a message from a node whose address corresponds to a certain hash value is identified and it is determined that the message was identified within a predetermined period from the identification of a previously identified message from a node whose address corresponds to same hash value, the counter is incremented by 1.

The address replacement module may be configured to determine if an address included in an identified message should be replaced in accordance with the timestamp and/or in accordance with the counter associated with the hash value which corresponds to the address of the node from which the address was received. The address replacement module will reset the counter associated with a certain hash value if the counter exceeds a predefined value. In addition, when a message from a node whose address corresponds to the hash value was allowed to proceed to its original destination, for example, to an external node - the counter will be reset as well. In accordance with one exemplary scenario, if a certain node repeatedly generates messages to enable it to exchange content with another node, and LiteLoad seems to have failed to provide alternative destinations to enable the requested content exchange, the node should be allowed to make the requested connection.

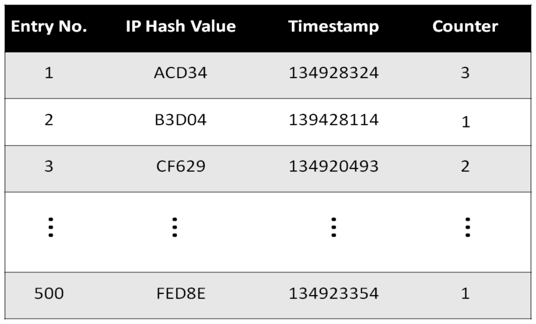
4.2.5 Links from External Entities

Notice in Figure 1 to the message E, which arrives from node *P7*. This client is operating a P2P application that tries to connect to a supernode that resides in ISP *X* network. As in previous cases, here again we intercept the session initiation message and filter it. Our interest in to allow a limited access of external peers to ISP*X* since we're interested in importing new content that is not shared already by local peers.

4.2.6 Session Initiation Message Detection

We recognize session initiation messages per protocol according to the header of the message. We were able to do so for current popular protocols such as BitTorent and EDonkey as well as older ones such as Fasttrack(KaZaA) and iMesh. Since a session initiation message is the first message being sent from the peer to a supernode, it's not encrypted. Our solution is applicable for encrypted protocols since we only use the session initiation messages for extracting the address of the supernodes. In case where the session initiation message is encrypted our solution will not work; However, this will only be possible in cases where encryption keys were embedded in the client software prior to initiating communication with the supernode.

We also use session initiation messages to learn about the addresses of internal supernodes. If the destination address of such filtered message is internal, we store it in a table or a pool of potential internal supernodes addresses.



Entry No.	IP Hash Value	Timestamp	Counter
1	ACD34	134928324	3
2	B3D04	139428114	1
3	CF629	134920493	2
⋮	⋮	⋮	⋮
500	FED8E	134923354	1

Figure 2. LiteLoad Hash Table

5. Results

5.1 Simulation

We simulated LiteLoad for the most popular P2P supernode based architecture. One interesting issue to check was LiteLoad's performance on various amount of files compared to a normal supernode network.

On each iteration of our simulation, a randomly chosen client selects a file to be downloaded. The requested file is being selected according to a long tailed power law distribution such that some files are more popular than others thus have better change to be elected by the client. We constructed 2000 peer objects, that are distributed randomly in 10 different zones. We changed the amount of total files in the system for each simulation run ranging from 1000 to 15000 files. The files were logically distributed randomly between the zones (each file object was tagged for a specific zone so we can later let peers select "local" files to be downloaded by preference). We created 50 super nodes and distributed them randomly among the zones. Prior to running time, each peer selected 100 files that are shared by it according to the same long tailed distribution mentioned above. The simulation stops after exactly 1000 iterations and informs how many peers were able to find their requested file in the supernode they were connected to.

The only difference here between the Normal mode and the LiteLoad mode is that in the normal mode each peer selects the supernode randomly and in the LiteLoad mode a supernode is chosen randomly out of the set of supernodes that share the same zone as the client peer.

In Figure 3 it can be seen that as the number of files in the system becomes larger, the normal supernode based protocols fail to supply the requested files. Notice that in this simulation if in the case of LiteLoad the peer couldn't find the requested file in his connected supernode, it reports it as a failure and the simulation continue to the next iteration. In fact, LiteLoad can perform even better if we allow it to redirect again to a different supernode after failure.

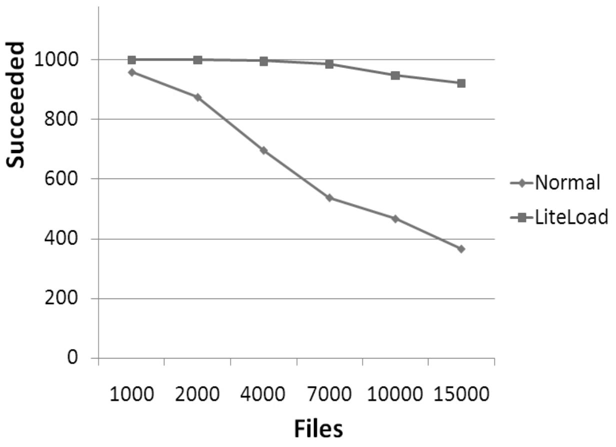


Figure 3. LiteLoad Simulation - different files count

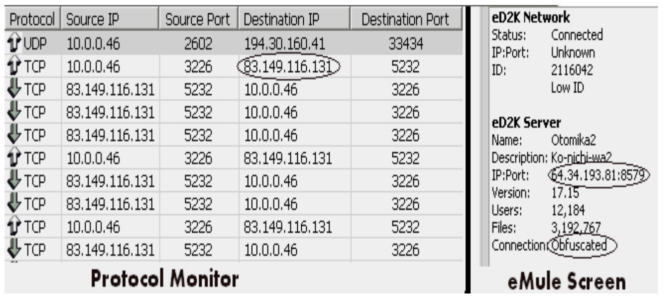
5.2 Existing Protocols Experiments

Though the concept of LiteLoad is fairly intuitive, we wanted to test the feasibility of the destination address redirection on existing protocols in general, and for encrypted protocols in particular. Therefore we implemented portions of the protocol - mainly the redirection mechanism by altering a Socks proxy server. This way we could monitor and alter messages as if LiteLoad was residing in the ISP's network.

We started with KaZaA. We installed three clients on three separate machines, denoted as A, B and C. We allowed B to be configured as a supernode. A was configured to transfer its messages through a local Socks proxy. We first tested KaZaA and were able to redirect a client to B instead of an external address it tried to connect to. In order to make sure that A was not only connected to B but also usable, we shared a file on C and redirected it to B as well. A was able to find this file and downloaded it successfully.

Then we advanced into BitTorrent. We allowed the proxy to alter the addresses that are served by the tracker (we let our client connect to an external tracker that was not under our control) and this way we were able to force a download from another local BitTorrent client.

Recently we adapted the Socks proxy to support eDonkey/eMule. eMule behaves similar to other supernode based networks like KaZaA. We set our client to support obfuscated-only connections, which is the encrypted version of eMule, and similarly to what we did with KaZaA we redirected the session initiation message to a different server. We were able to perform well under redirected connections and could download content. Notice in Figure 4 that while we requested on eMule to connect to the server on 64.34.193.81:8579, our redirection made it possible to connect to 83.149.116.131:5232 under an obfuscated connection!



Protocol	Source IP	Source Port	Destination IP	Destination Port
UDP	10.0.0.46	2602	194.30.160.41	33434
TCP	10.0.0.46	3226	83.149.116.131	5232
TCP	83.149.116.131	5232	10.0.0.46	3226
TCP	83.149.116.131	5232	10.0.0.46	3226
TCP	83.149.116.131	5232	10.0.0.46	3226
TCP	10.0.0.46	3226	83.149.116.131	5232
TCP	83.149.116.131	5232	10.0.0.46	3226
TCP	83.149.116.131	5232	10.0.0.46	3226
TCP	10.0.0.46	3226	83.149.116.131	5232
TCP	83.149.116.131	5232	10.0.0.46	3226

```
eD2K Network
Status:      Connected
IP:Port:     Unknown
ID:          2116042
             Low ID

eD2K Server
Name:        Otomika2
Description: Ko-nichi-wa2
IP:Port:     64.34.193.81:8579
Version:     17.15
Users:       12,184
Files:       3,192,767
Connection:  Obfuscated
```

Figure 4. LiteLoad eMule Experiment on Obfuscated Protocol

6. Conclusions

In this paper we presented Liteload - a solution for localizing P2P protocols in ISPs that is unaware of the content being transferred. The concept was tested in a simulation that showed a fundamental advantage.

We demonstrated a proof of concept on existing popular protocols such as eMule/eDonkey and BitTorrent. We successfully tested it on an obfuscated version of eMule as well. As LiteLoad only examines session initiation message, it is fully unaware of the content that is being transferred between peers and since it does not perform indexing on the content as well, our solution enables ISPs to operate it with confidence.

In the future we plan to explore adjacent fields and problems to enrich our solution and implement it on a large scale ISP. We will also examine performance issues on random supernode selection as opposed to selecting peers according to their network behavior patterns. In addition we will examine how we can further improve the tolerance of the system to different behavioral patterns of users.

References

[1] Allot web site. http://www.cachelogic.com.

[2] Azureus bad isps wiki web page. http://www.azureuswiki.com/index.php/bad_isps.

[3] Cachelogic web site. http://www.cachelogic.com.

[4] Cisco's pcube web site. http://www.p-cube.com/.

[5] Dailytech story about bell simpatico traffic shaping. http://www.dailytech.com/more+isp+ confess+ we+ throttle+ p2p+ traffic/article9544.htm.

[6] Dailytech story about comcast and peer to peer. http://www.dailytech.com/comcast+ screws+ with+ filesharing+ traffic/article9337.htm.

[7] Ipoque web site. http://www.ipoque.com.

[8] Joltid web site. http://www.joltid.com.

[9] Packeteer packetshaper web site. http://www.packeteer.com/products/packetshaper/.

[10] Peerapp measurments on p2p. http://www.peerapp.com/solutions- managing- transit-link-growth.aspx.

[11] Peerapp web site. http://www.peerapp.com.

[12] Sandvine incorporated web site. http://www.sandvine.com.

[13] V. Aggarwal, A. Feldmann, and C. Scheideler. Can isps and p2p users cooperate for improved performance? *SIGCOMM Comput. Commun. Rev.*, 37(3):29–40, 2007.

[14] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, volume 37, 5 of *Operating Systems Review*, pages 314–329, New York, Oct. 19–22 2003. ACM Press.

[15] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should isps fear peer-assisted content distribution? *In IMC*, 2005.

[16] N. Leibowitza, A. Bergman, R. BenShaul, and A. Shavit. Are file swapping networks cacheable? characterizing p2p traffic. *In WCW'02*, 2002.

[17] G. Shen, Y. Wang, Y. Xiong, B. Zhao, and Z. Zhang. Hptp: Relieving the tension between isps and p2p. *In IPTPS*, 2007.