Utilizing Traffic & User Behavior Patterns in Peer-to-Peer Networks

Thesis for the degree of

DOCTOR of PHILOSOPHY

by

Shay Horovitz

SUBMITTED TO THE SENATE OF

THE HEBREW UNIVERSITY OF JERUSALEM

February 2011.

This work was carried out under the supervision of

Prof. Danny Dolev

Acknowledgements

The writing of this dissertation has been an outstanding journey, walking through endless enjoyable challenges, particularly enjoyable as a result of the interaction with my supervisor, colleagues, friends and family.

I feel very privileged to have worked with my supervisor, Prof. Danny Dolev. Words cannot express my deep appreciation for all his contributions of time around the clock, wisdom, endless patience and funding to make my research experience productive and stimulating. Without his guidance as a great mentor, this work would not have been possible.

I would also like to thank all the members of the Distributed Algorithms, Networking and Secure Systems Group (DANSS) at the Hebrew University, specifically Dr. Tal Anker for his time and support.

I want to express my gratitude to my friend, Yehoshua Sapir for his constructive comments.

Without the love and support of my family, this would have been a very hard journey. I thank my parents for their love and unconditional support over the years and my brothers, Shlomi that always cared for my progress and Itsik for endless hours of discussions and comments that practically made him my second mentor during my PhD. I am truly lucky to have such a great family.

Last but not least, I am very grateful to my beautiful wife, Precillia, for her love and patience during my PhD studies.

Abstract

Peer-to-Peer (P2P) networks gained their reputation throughout the last decade thanks for breaking down communication barriers while enabling media content exchange in a click of a button. Yet, the complexity of P2P architecture had an unfortunate side-effect in the form of fluctuations in download rate and service fairness issues for the users while becoming a heavy financial burden for Internet service providers.

This work contribution is in increasing the level of understanding of P2P behavioral aspects and presenting novel solutions for key challenges by utilizing traffic, user and application behavior patterns in P2P networks.

The first part of this thesis present how modification of traffic behavior patterns can help service providers with localizing popular P2P networks including encrypted ones, without raising the legal disputes of caching solutions or compromising the QoS as in traffic shaping solutions. Further, the traffic behavior patterns of P2P file sharing networks are being modified in order to gain stable download rates and allowing existing P2P networks to efficiently utilize extremely weak or unstable sources.

The second part addresses the effects of user behavior on the performance of P2P file sharing and streaming networks. Using supervised machine learning algorithms the solution presented can learn the behavior patterns of the user and predict the effects of the user on the throughput of his machine, increasing the download rate of P2P file sharing networks and preventing hiccups in P2P streaming networks.

The final part discusses how application behavior patterns can help with predicting excessive resource consumption of applications. Unsupervised machine learning algorithms are being used to learn the behavior patterns of each application by utilizing network, processor and file system events. Finally, a solution for identifying encrypted protocols such as Skype behaviorally is presented, without using traditional port or payload based deep packet inspection techniques.

Contents

Chapter 1

Introduction

P2P technology earned its fame throughout the last decade thanks to the wide deployment of P2P file sharing applications over the Internet in the late 1990s. Among the early releases, the popular ones were Napster, Scour Exchange, iMesh and Gnutella which were followed by improved designs such as KaZaA, eDonkey and BitTorrent. Following the increased popularity of online video content, new designs of P2P streaming networks were proposed by Joost, PPLive and others. In parallel, the research community innovated some promising designs for meeting the major challenges that relate to P2P networks - mainly with the Lookup problem [95, 90, 107, 87] but also in security, scalability and performance. The potential of P2P to the end user in a P2P network is obvious - the ability to receive content (in some cases free of charge) easily, backed by an efficient search and an active community that continuously update the shared content.

1.1 QoS problems in P2P Networks

While the above seems promising, recent measurements of broadband usage patterns in ISPs reveal a surprising rising trend that should concern the P2P research community: new server based services are growing in traffic at the expense of P2P traffic [30, 77]. While according to a recent report by Cisco [11], P2P traffic is predicted to double to more than 7 Petabytes per month by 2014; other types of file-sharing services such as RapidShare and MegaUpload are expected to grow explosively, increasing nearly eightfold to 4 Petabytes per month by 2014.

As claimed in [7], subscribers are increasingly turning to alternatives such as File Hosting web sites like RapidShare and MegaUpload, since they enable much faster download speed compared to P2P networks. BitTorrent [50] for example - the most popular P2P protocol, suffers from unstable download rates and hardly exploits the available download capacity [44, 37]. Related work in this field can be found in [54, 42, 26, 104, 75, 43, 48].

But not only file sharing P2P services suffer from stability and performance. Popular implementations of P2P based streaming networks in the industry such as Joost [24] and PPLive [29] were reported to often suffer from a variety of QoS problems [100, 13, 15, 5] like broken streams, streaming audio/video hiccups, substantial latency and major delays in live broadcasts; even when the network's policy allowed extremely long latencies of up to 2 minutes [100, 65], users were still faced the above problems. Related architectures for P2P streaming can be found in [97, 56, 53, 47, 88, 62]. In contrast, non P2P video services such as NetFlix [27] are on the rise according to a recent traffic report by Sandvine [30].

Studies have shown that the major factor that has direct impact on QoS in P2P networks is the behavior of users at the source peers [56, 88] - taking occasional actions that heavily use bandwidth such as sending Email, online games, running other P2P applications in parallel or even terminating the process of the P2P network while it is streaming.

The above problems put P2P technologies in question for commercial system designers, due to QOS problems. As most P2P systems already run a best effort approach by prioritizing peers with minimized infrastructure problems like delay and packet loss, they still miss a key factor in degrading P2P performance - the user behavior. In addition, this approach is blind to a large number of weak sources that remain unutilized, while a small group of strong sources are heavily exploited and overused [73].

Thus there is a need for a solution that will address the problem of user behavior in P2P streaming networks and provide a mechanism that is able to absorb the instability of source peers and allow stable throughput at the client side; this may be done by modifying the traffic patterns that are used in P2P networks as presented with Collabory in Chapter 2 or by learning the user behavior patterns and predict upcoming problems in advance as presented with Collabrium in

Chapter 3.

1.2 The Cost of P2P for ISPs

P2P brought some more complex traffic patterns to the internet and along with them, some economic effects. 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.

Though many ISPs still deny employing solutions for that problem, Canada based Bell Simpatico has confessed [1] to using "traffic management" on heavy users during peak hours. In addition, a testing [2] 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. According to the list of "Bad ISPs" [33] that is maintained by the Bittorrent community, some 120 ISPs worldwide employed traffic shaping for either limiting or blocking P2P protocols in 2010, using traffic-shaping solutions such as those offered by Allot [16], Cisco (PCube) [19] and Packeteer [18].

Several studies [60, 76] showed that bandwidth savings of the order of 60% are achievable by exploiting traffic locality in P2P. Thus, solutions that were offered by CacheLogic [10], PeerApp [14] and Joltid [12] 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 [79]. 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, caching seems to be losing its grip in the market.

This motivates the need for a new approach that makes it possible to localize P2P protocols without being aware of the requested and transferred content. Thus, it should neither employ caching of content nor maintain tables where one can deduce that specific portions of a given content are being shared by specific user as it contradicts the interests of the ISP. In addition, it

should also support encrypted protocols, as caching or any other solutions that perform reverse engineering of the protocol cannot handle encrypted protocols. This may be done by modifying the traffic patterns of P2P protocols at the ISP as presented with LiteLoad in Chapter 2.

1.3 Behavior Patterns

This work contribution is in increasing the level of understanding of P2P behavioral aspects and presenting novel solutions for the key challenges mentioned above by utilizing **traffic**, **user** and **application** behavior patterns in P2P networks. Following, each of the different behavior patterns is being addressed.

1.3.1 Traffic Behavior Patterns

Chapter 2 analyzes traffic behavioral patterns that are typical in P2P networks and modify these patterns for selected applications. In order to modify the existing behavioral traffic patterns of P2P protocols LiteLoad [69] was created - a solution that discerns patterns of traffic 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. LiteLoad utilizes the typical traffic patterns of modern Super Node based P2P networks and modifies the links of content flow. 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 show the potential of the solution and a proof of concept of the key technology is demonstrated on popular protocols, including encrypted ones.

Another interesting traffic pattern in P2P file sharing networks is in the flow between source peers and clients, where in most modern P2P networks a client (not surprisingly) downloads content directly from source peers. Common peer-to-peer (P2P) file sharing clients usually download at an unstable rate and hardly exploit the available bandwidth offered by low rate sources. The characteristic fluctuational throughput of the source peers might be caused by user behavior factors such as running other bandwidth consuming tasks, throttling of download speed

by P2P software or even termination of the source.

For addressing the above problem, Collabory [68] is proposed - a software agent solution for stabilizing and accelerating the download speed rate in existing P2P networks. A new role is introduced: "Feeders" - special kind of Helper Peers [72] that modify the traffic patterns between clients and sources in P2P networks, by collaboratively aggregating the traffic from multiple sources into a single, stable stream served to the downloading peer. The tests show that the solution utilizes source nodes with an extremely low and unstable throughput without reducing the download rate of the downloading peer. Upgraded & stabilized throughput is demonstrated on eMule.

1.3.2 User Behavior Patterns

Emerging large scale file sharing Internet applications base their infrastructure on P2P technology. Yet, the characteristic fluctuational throughput of source peers affect the QOS of such applications which might be reflected by a reduced download rate in file sharing or even worse - annoying freezes in a streaming service. A significant factor for the unstable supply of source peers is the behavior of the user that controls other processes running on the source peer that consume bandwidth resources.

If one can learn the behavioral patterns of the user that runs various applications that consume bandwidth, it will be possible to utilize this information for building better P2P networks. Chapter 3 presents Collabrium [70] - a collaborative solution that employs a machine learning approach to actively predict the load in the uplink of source peers and alert their clients to replace their source. Experiments demonstrated successful predictions of upcoming loads and Collabrium learned the behavior of popular heavy bandwidth consuming protocols such as eMule & BitTorrent correctly with no prior knowledge.

Yet, the instability of throughput in P2P networks is not limited only to file sharing applications; in theory, peer-to-peer (P2P) based streaming designs and simulations provide a promising alternative to server-based streaming systems both in cost and scalability. In practice however, implementations of P2P based IPTV and VOD services failed to provide a satisfying QoS as

the characteristic fluctuational throughput of a peer's uplink leads to frequent annoying hiccups, substantial delays and latency for those who download from it. Here again, the user's selection of applications that require network bandwidth resources, play a significant factor for the unstable throughput of peers' uplink with processes running on the source peer that consume bandwidth resources.

Following Collabrium, Maxtream [71] is being presented - an adaptation of our machine learning based solution for streaming peers, coordinating source peers exchanges between peers that suffer from buffer underrun and peers that enjoy satisfactory buffer size for coping with future problems.

1.3.3 Application Behavior Patterns

While user behavior has its implications on the final traffic by controlling the applications installed in his own machine, the actual packets being sent are subject to the commands of each application. If it's possible to learn the behavioral patterns of each application utilizing its own commands, it will also be possible to predict the application's behavior with higher accuracy.

A running application can be seen as an entity that supports various activities - such as sending email, downloading a file, browsing a web page etc. In each activity, the application calls a sequence of actions - for example if the activity is sending an attachment, then among the first actions one will find an action that reads the attachment file into the memory; later in the sequence of actions one will find an action that sends fragments of that file as network packets. The Application Behavioral Patterns of a specific application is the set of sequences that reflect its different activities.

Chapter 4 presents Seekuence - a solution that using unsupervised machine learning algorithms, learns the behavioral patterns of specific applications by analyzing the sequence of actions taken by application events such as packets sent, CPU usage and file system events. First, the effect of the application on the consumption of resources that can be beneficial for applications such as cloud computing is analyzed; then a case study using Seekuence for inspecting encrypted protocols such as Skype by utilizing application behavioral patterns is presented without using

traditional port or payload based deep packet inspection (DPI) techniques.

1.4 Dissertation Outline

The material discussed in this thesis covers the following papers: [69, 68, 70, 71, 72] and a patent [67]. The rest of this dissertation is organized as follows: Chapter 2 discusses the effect of Traffic Behavior Patterns on the cost of P2P networks for ISPs and how to modify these patterns with LiteLoad [69]. Following, the new notion of Feeders is being introduced as an evolution of Helper Peers [72] and their contribution for stabilizing the download rate in P2P file sharing clients with Collabory [68]. Chapter 3 presents how supervised machine learning algorithms can predict the effect of User Behavior in P2P file sharing networks with Collabrium [70] and in P2P streaming networks with Maxtream [71]. Chapter 4 discusses how Seekuence can help in predicting excessive resource consumption by P2P applications by utilizing unsupervised machine learning algorithms, and presents a solution for identifying encrypted protocols behaviorally using the Application Behavioral Patterns. Chapter 5 presents our conclusions and issues for future work.

Chapter 2

Traffic Behavioral Patterns

2.1 Modifying P2P Traffic Behavior

In today's extensive worldwide Internet traffic, between 40% to 70% [64, 28, 30] of network congestion is caused by Peer to Peer sessions. P2P brought some more complex traffic patterns to the internet and along with them, some economic effects. 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.

In this chapter, traffic behavioral patterns will be analyzed and modified for selected applications. In order to modify the existing behavioral traffic patterns of P2P protocols LiteLoad was created - a solution that discerns patterns of traffic 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. LiteLoad utilizes the typical traffic patterns of modern Super Node based P2P networks and modifies the links of content flow. 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 express the potential of the solution and a proof of concept of the key technology is demonstrated on popular protocols, including encrypted ones.

Another interesting traffic pattern in P2P file sharing networks is in the flow between source peers and clients, where in most modern P2P networks a client (not surprisingly) downloads content directly from source peers. Common peer-to-peer (P2P) file sharing clients usually download at an unstable rate and hardly exploit the available bandwidth offered by low rate sources. The characteristic fluctuational throughput of the source peers might be caused by user behavior factors such as running other bandwidth consuming tasks, throttling of download speed by P2P software or even termination of the source.

For addressing the above problem, Collabory is being proposed - a software agent solution for stabilizing and accelerating the download speed rate in existing P2P networks. A new role: "Feeders" is being introduced - peers that modify the traffic patterns between clients and sources in P2P networks, by collaboratively aggregating the downloads from multiple sources into a single, stable stream served to the downloading peer. It is shown that the solution utilizes source nodes with an extremely low and unstable throughput without reducing the download rate of the downloading peer. Upgraded & stabilized throughput is demonstrated on eMule.

2.2 LiteLoad 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 were believed to be responsible for approximately 60% [28, 64] of all Internet traffic. A Recent study shows that despite the rise of alternative non-P2P media services such as NetFlix [27] in the US, P2P is still responsible about 60% [30] of Upstream.

Internet Service Providers (ISPs) typically pay for external links (links with nodes located outside the specified portion of the ISP'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 ISP'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 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 or required to store or cache any data which is associated with or including reference to the content. In addition, support of encrypted P2P protocols is required.

LiteLoad is being presented as 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, a proof-of-concept prototype was built 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 traffic patterns of P2P protocols and accordingly apply rules that influence the behavior of such protocols for creating a localized network.

2.3 LiteLoad 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. [35] 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 [93] 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 [10], PeerApp [14] and Joltid [12] 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 [79]. 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 [60] 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 [31]. 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.

2.4 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 [1] 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 [2] 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 ISPs.

A survey from traffic-management company Ipoque [23] 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). In China for example,P2P applications account for more than 50% of the Internet traffic in the daytime and 90% at night for China Telecom, the largest ISP in China [3]. According to a recent study about the US market, P2P is still responsible for about 60% [30] of the upstream.

For better understanding the size of the problem and learn the required properties for a feasible solution, several ISPs were visited. 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 [60, 76] 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 [16], Cisco (PCube) [19] and Packeteer [18]. 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 [17] of ISPs that use traffic shaping and provides a set of simple steps how to avoid that.

2.5 LiteLoad Architecture

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.

2.5.1 Traffic 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, let's examine the communications pattern of a browser that browses a simple HTML page that contains an image. When monitoring the network activity it can be shown 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 the following sequence is introduced:

1. Client asks for index.html;

2. Client asks for index.html;

3. Server returns index.html;

One can see 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, one can consider it as normal. By employing similar techniques one can test P2P protocols and learn their behavior on different scenarios. Once the protocol behavior is known, it may be possible to trick it to perform various actions, depending on the protocol and topology. In this work these techniques are demonstrated and it is shown how they can support us in localizing P2P protocols.

2.5.2 Structure

Let us begin by first describing 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. The first message that is being sent to a potential supernode by the client is being termed 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 ISP'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 that

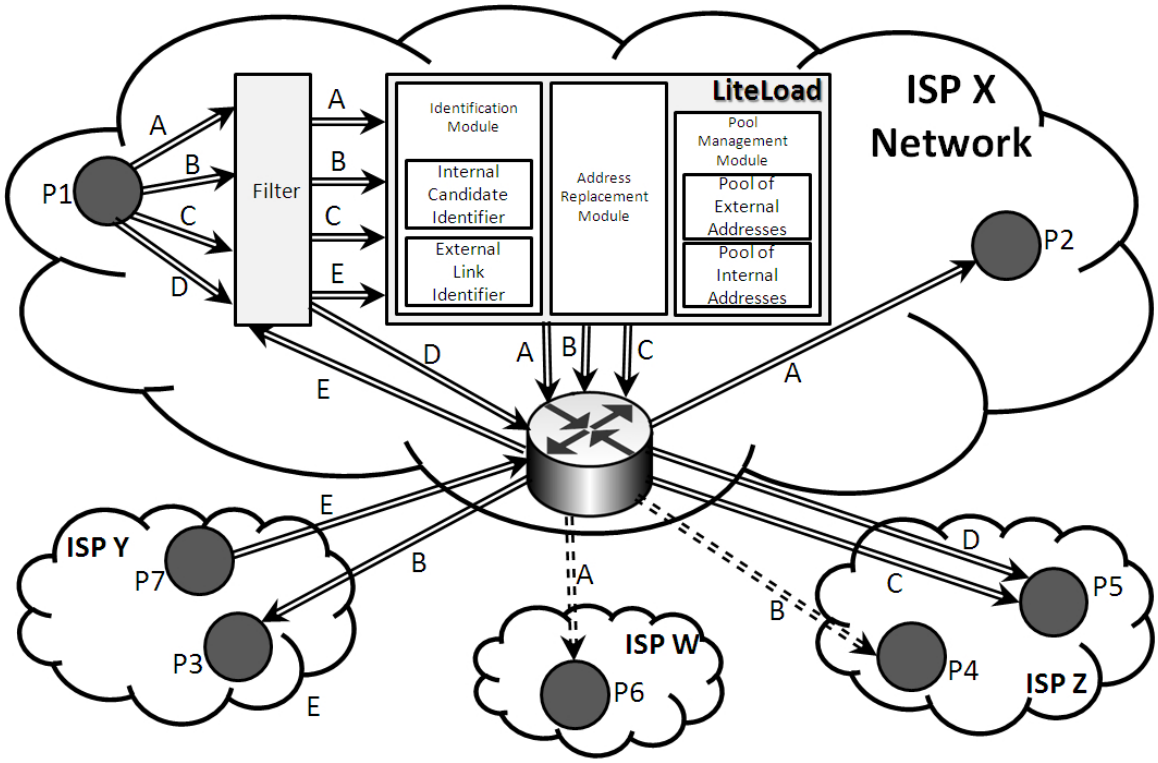

Figure 2.1: LiteLoad Architecture

was chosen to replace the address is content insensitive as only the header of a session initiation message was modified. Moreover, the actual content in the messages was not intercepted.

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.

Address Replacement Module

Referring to Figure 2.1: *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 2.1, 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.

Address Pools

LiteLoad includes at least one pool of replacement addresses. In the sample presented in Figure 2.1, 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 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.

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.

Access Hash Table

Figure 2.2 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.

Links from External Entities

Notice in Figure 2.1 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

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

Figure 2.2: LiteLoad Hash Table

cases, here again the session initiation message is intercepted and filtered. Our interest is to allow a limited access of external peers to ISPX since we're interested in importing new content that is not shared already by local peers.

Session Initiation Message Detection

Session initiation messages are being recognized per protocol according to the header of the message. It was possible 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. The solution is applicable for encrypted protocols since only session initiation messages are being used for extracting the address of the supernodes. In case where the session initiation message is encrypted the 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.

In addition, Session initiation messages were used to learn about the addresses of internal supernodes. If the destination address of such filtered message is internal, it is being stored in a table or a pool of potential internal supernodes addresses.

2.6 Testing LiteLoad

2.6.1 Simulation

LiteLoad was simulated 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. 2000 peer objects were constructed, that are distributed randomly in 10 different zones. The amount of total files in the system was changed 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 later peers can select "local" files to be downloaded by preference). 50 super nodes were created 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 2.3 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 allowed to redirect again to a different supernode after failure.

LiteLoad had higher success rate in Figure 2.3 since it localizes P2P connections - thus improving the chances that the peers residing inside the same ISP will find content that is being shared locally. Geography plays a significant role in content consumption, due to the common

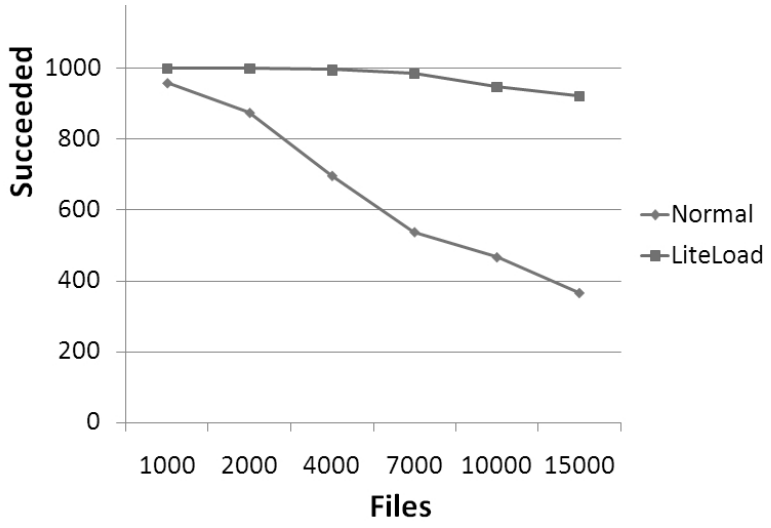

Figure 2.3: LiteLoad Simulation - different files count

language and culture.

2.6.2 Existing Protocols Experiments

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

Starting with KaZaA, three clients were installed on three separate machines, denoted as A, B and C. B was allowed to be configured as a supernode. A was configured to transfer its messages through a local Socks proxy. KaZaA was first tested and it was possible 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, a file was shared on C and redirected it to B as well. A was able to find this file and downloaded it successfully.

Next was testing BitTorrent: by allowing the proxy to alter the addresses that are served by the tracker (the client is allowed to connect to an external tracker that was not under control)

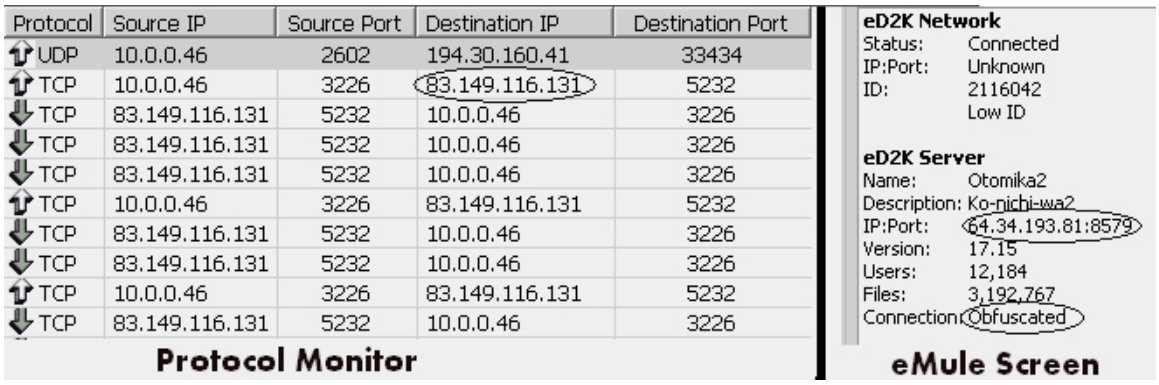

Figure 2.4: LiteLoad eMule Experiment on Obfuscated Protocol

finally it was possible to force a download from another local BitTorrent client.

Next the Socks proxy was adapted to support eDonkey/eMule. eMule behaves similar to other supernode based networks like KaZaA. The client was set to support obfuscated-only connections, which is the encrypted version of eMule, and similarly to what was donewith KaZaA the session initiation message was redirected to a different server. eMule was able to perform well under redirected connections and could download content. Notice in Figure 2.4 that while eMule was requested to connect to the server on 64.34.193.81:8579, the redirection made it possible to connect to 83.149.116.131:5232 under an obfuscated connection!

2.7 Collabory Introduction

Following the increased popularity of P2P, came a corresponding increase in the amount of research on the lookup problem. Yet, the common method being used for the data transfer between peers still relies on traditional client-server techniques and ignore the unique characters of P2P networks. A significant difference in download rate stability was noticed on the client side as it downloads from P2P peers compared to downloading from a server. In a home machine the throughput of the internet connection is directly affected by the user's behavior - running different bandwidth and cpu demanding tasks in an unpredictable manner resulting in frequently sudden changes in the effective throughput and instability in the download rate.

In the rest of this chapter the problem of throughput stability in P2P networks is being addressed. The causes for such instability is being discussed and a design that can be easily adapted to any P2P network is offered for creating a much more stable download rate and respectively gain higher throughput. Instability of the download rate stems from several reasons, which is being categorized as network related, P2P network design and user behavior aspects.

Network infrastructure aspects that affect the stability of a peer are not under the control of the user. Large delay variation (jitter), excessive packet delays and round trip time reduce the ability to ensure a stable connection. Packet loss might also contribute to the download rate instability especially between different regions. In addition, ISP enforced policy controlled by traffic shapers downgrade P2P download rate at peak hours [63].

P2P network design aspects are connected to how the client searches for potential peers. In many networks, ping RTT measurements are used as a metric for better sources. However our experiments showed that this technique does not produce a stable behavior. In addition, the problem was addressed also in [78] where it was shown that latency measurements are a poor predictor of P2P TCP throughput. Further, if peers with maximal upload rate were actually found, it does not guarantee that those peers are stable and the resulting throughput to be higher in the long term. Moreover, when such a source is heavily used, the disruption in case of a failure during the download session is much higher. In addition, P2P networks embed fairness rules for preventing free riders [106,57]. This might affect the stability during the download phase as there

is a certain ratio between the download rate and the amount of data a source uploads.

Yet, the aspect that was found having the greatest impact on download rate stability is the behavior of users at source peers. More specifically, actions that the user of the uploading peer machine occasionally takes might directly affect the upload rate of the machine. The most obvious occurrence is the case where the user at the source peer invokes applications that heavily use bandwidth such as Email clients, online games or other P2P applications. By doing so, the bandwidth available for the client connected to that machine may be drastically reduced and becomes significantly unstable.

Following, Collabory is being presented - a software agent collaboration tool utilizing helper peers for stabilizing P2P clients' download rates. Helper peers are peers that voluntarily participate in the file download by pre-fetching and caching parts of the file for the client. The idea of utilizing helper peers was mentioned in [103, 59, 86, 102], however, they all offered using helper peers (in addition to normal source peers) for bypassing the limitations of fairness rules that are embedded in P2P networks [50, 21]. Moreover, the helpers selection criteria in these papers does not guarantee better stability compared to the stability of normal source peers i.e., the fluctuational rate of the sources and helpers is still reflected in the rate of the client. Therefore, the client is still exposed to stability problems due to network infrastructure, P2P network design and user behavior aspects.

Existing P2P protocols tend to ignore source peers that have relatively low bandwidth to offer. This dramatically reduces the amount of available sources and eventually the downloading peer ends up with insufficient available sources to maximize its throughput. Our solution enables utilizing extremely unstable peers and low bandwidth peers for maximizing the final throughput of the downloading client without being exposed to the sources instability. The above problems motivate the need for a new data transfer approach, adjusted to the P2P network characteristics and the behavior pattern of end users.

Collabory employs a helper-like architecture. Unlike previous works, it intentionally selects the helpers to be optimal for availability and throughput stability *with the client* by constantly measuring stability factors. Let us term such helpers as *feeder* peers. The Feeders negotiate with

potential source peers and aggregate the downloads from multiple unstable sources into a single, stable stream served to the downloading peer. The Feeders are employed exclusively as a means of delivering data to the client.

For estimating the level of fitness of a potential feeder, first the availability of a feeder is being optimized by matching a client with feeders that share similar online periods yet dissimilar in activity periods. This way the chance that feeders will disconnect intentionally while the client is downloading is reduced. Among the feeders that match, those that are most predictable are preferred - i.e., have repetitive overlapping online sessions with high probability. Furthermore, one should strive to match feeders that have good network connectivity with the client. The jitter, RTT and packet losses are being measured constantly and accordingly selecting those with the best connectivity measurements. In addition, a high priority on the effect of user behavior on the stability of a potential feeder is being taken. Thus, feeders with a fair amount of predictable spare bandwidth are preferred and if possible, low CPU usage. The incentive for serving as a feeder is by contributing machine resources in return to a higher throughput when one wishes to download.

To evaluate the concept, a test suite was build to monitor the download rate and its stability given different scenarios of network infrastructure problems as well as user behavior patterns at the source peers. The results of our experiments confirmed the following contributions of Collabory:

1. Ensuring a stable download rate resistant to source throughput instability with higher probability than in current techniques;
2. Allowing weak - low bandwidth sources to participate in the download process without compromising the client's download rate stability, thus increasing the number of potential sources;
3. Ability to locate potential sources and measure their throughput without compromising the client's download rate stability.

The remainder of this chapter examines these issues both analytically and empirically and presents Collabory.

2.8 Helpers Peers

2.8.1 Adding Sources in File Sharing and Multicast

The concept of employing helper peers in a P2P networks was first proposed by Wong [103]. In his work, which was limited to file sharing based on a swarming mechanism, Wong offered to utilize the free upload capacity of a helper peer for the benefit of client peers, simply by joining helper peers to an existing swarm (see Figure 2.5).

The helper aim to upload each piece it downloads at least u times, where u is a heuristically predetermined number called upload factor. Thus, helpers can guarantee to upload more than they download and contribute to the system. To make sure each piece it downloads is uploaded at least u times, a helper keeps track of the number of times each piece has been uploaded and considers a piece unfulfilled if the piece has not been uploaded u times. The helper downloads a new piece when the number of unfulfilled pieces is below a certain predetermined limit. It's objective was to increase the total amount of available bandwidth in the P2P network, by voluntarily contributing the helper peer's bandwidth resources. It was shown that this strategy is wasteful because the longer a peer/helper stays in the system, the more pieces it will download, which is unnecessary for helpers to keep their upload bandwidth fully utilized [102].

It was also shown that the inherent assumption of sufficient altruism in the network without any incentives makes the approach impractical in real world environments [86]. While Wong presented a new mechanism for increasing the available bandwidth at the network level, the performance at the client's side was still in question as the proposed mechanisms did not address the problem of bandwidth stability of a helper peer - which is a major factor for the performance of the download process. In addition, the whole design is not generic but is based solely on swarming that is managed by a tracker; this limits the potential of the solution to BitTorrent based systems only.

Following Wong's work, Wang et. al. [102] proposed a mechanism where the helpers need to download only small portions of a file to be "busy" enough for serving other peers in the long term. This work is also limited to BitTorrent [50] protocol. Yet, it is claimed that the increased

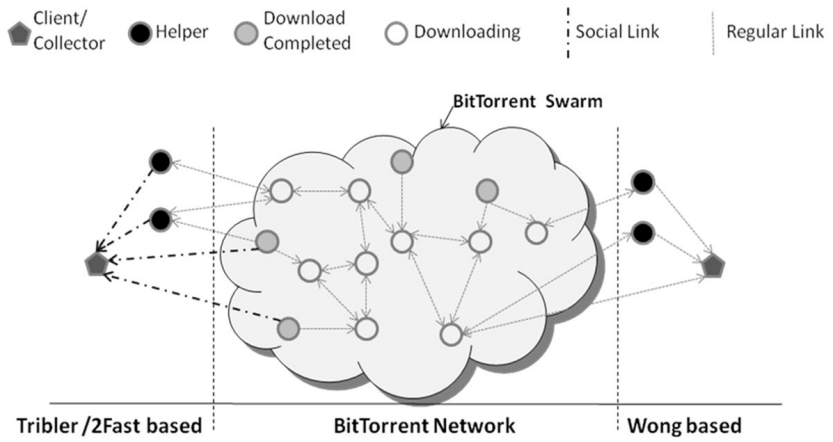

Figure 2.5: Wong and Tribler's additions to a BitTorrent swarm-based network

upload contribution only marginally improves download rates in BitTorrent [85]. In addition, it is considered that the network environment is homogeneous - where users have the same link capacities. This is clearly an unrealistic assumption given Internet's heterogeneity.

In a recent work [101] it is proposed to employ helper peers in a hybrid network for streaming video content at a speed that is higher than the average upload bandwidth of peers. The authors discuss the term helpers as peers that are not participating in the multicast. Unlike the case of file sharing where users tend to leave their machine running for predefined downloads, in streaming the user has no motivation for leaving the application up and running when not used for streaming. Other works that proposed similar ideas of using helpers for multicast are DynaPeer [94] and [52], where helpers take part in a collaboration process for a specific video stream that is managed by a virtual server.

In AntFarm [84], a design aiming for high throughput by viewing content distribution as a global optimization problem, Coordinators manage the system by issuing tokens, computing response curves, and determining bandwidth allocations while Seeders (that can be seen as helpers) expend their bandwidth to distribute blocks of files to peers. AntFarm depends on Coordinator servers, that are managed by the content distributor.

2.8.2 Social Helpers

In Tribler [86] it was proposed to associate the helpers' contribution with social phenomena such as friendship and trust. In their 2Fast file sharing protocol [59], a peer trying to download a file actively recruits its "friends", such as other peers in the same social network, to help exclusively with its download. 2Fast was originally offered to overcome the problem of free-riding in P2P networks. Peers from a social group that decide to participate in a cooperative download take one of two roles: they are either collectors or helpers. A collector is the peer that is interested in obtaining a complete copy of a particular file - like a typical client in a P2P network, and a helper is a peer that is recruited by a collector to assist in downloading that file. Both collector and helpers start downloading the file using the classical BitTorrent tit-for-tat and cooperative download extensions (see Figure 2.5). Before downloading, a helper asks the collector what chunk it should download. After downloading a file chunk, the helper sends the chunk to the collector without requesting anything in return.

In addition to receiving file chunks from its helpers, the collector also optimizes its download performance by dynamically selecting the best available data source from the set of helpers and other peers in the Bittorrent network. Helpers give priority to collector requests and are therefore preferred as data sources. Specifically, a peer will assign a list of pieces to obtain for each of its helpers. These are the pieces that it has not started downloading. The helpers will try to obtain these pieces just like regular leechers and upload these pieces only to the peer they are helping. In such a scheme, peers with more friends can indeed benefit greatly and enjoy a much reduced file download time. However, it was shown that the constraint that helpers only aim to help a single peer requires the helpers to download much more than necessary to remain helpful to this peer [102].

The fact that the help is served only by social linked helpers is a limit for the success of the solution as some peers might not have any social links and some might have, but the "friends" are not online or running the Tribler client when required. As for Wong's work, Tribler did not address the problem of bandwidth stability of helper peers. Again, this work's contribution is also limited to BitTorrent-like swarming architectures.

In between Wong's work and Tribler, Guo et al. [61] proposed a different mechanism of inter-torrent collaboration, where peers may download pieces of a file in which they are not interested in exchange for pieces of a file they want to download. Yet, it was shown that this approach will unnecessarily provide any performance gain [101].

The main contribution of the above mentioned works is in enabling a multicast download system which circumvents bandwidth asymmetry restrictions by recognising peers for their contribution of idle bandwidth, thus - increasing service availability.

2.8.3 Fairness, Free-Riding and Last Chunks

In addition to the anti free-riding solution that was proposed in 2Fast and Tribler, it was shown [74] that pairs of peers can collaborate as helpers for the benefit of fairness and anti free-riding. Yet, this work assumes that the collaboration is possible only between peers that have similar upload bandwidth. This requirement is problematic as the available upload bandwidth in a typical peer is subject to change over time. [91] offered a way to prevent the *last chunk problem* by cooperating between source peers.

2.8.4 Key Lookup

In P-Grid [51] - an index structure for P2P systems that is based on the concept of Chord, entries are owned by peers within strict bounds. The peers that do not take part in the structure are termed in the paper as helper peers; those peers are obliged to "help" a peer that is already in the ring by managing some part of the range indexed by it - this is done for load balancing of requests in a P2P ring structure. This resembles the previous mentioned works in the idea that a peer assists other peers even though it does not ask for a service for its user.

The above papers offered techniques to increase the download rate of a peer by either bypassing fairness rules or by sharing fairness credit among a social group of peers. However, the helpers selection criteria in these papers do not guarantee better stability compared to the stability of normal source peers. Thus, the client is still exposed to stability problems due to network infras-

tructure, P2P network design and user behavior aspects. Moreover, the above solutions do not address low rate sources, frequent joins and leaves and unstable sources - which are characteristic to P2P networks and ignoring them opens a substantial gap between theory and reality.

2.9 Throughput Stability in P2P

While performing some throughput tests for various P2P applications, a a major difference in download rate stability of the client when downloading from peers using P2P software was encountered compared to downloading from a file server.

In a typical P2P network, the user connects to the network and searches for a file using keywords. Then the P2P network returns a list of available files according to the keywords provided earlier. The user selects a file to be downloaded and in return, the P2P network provides a list of IP addresses, L, of peers that share portions of the requested file (this part is done automatically and is hidden from the user). Some P2P networks somewhat differ from the above description, yet provide a list of potential source peers. The part of the process that was described so far is also known as the Lookup problem. The Lookup problem was extensively researched and many publications presented innovative techniques for managing a directory of files in a P2P network [95, 90, 107, 87].

In this chapter the mechanisms for creating the list of sources L will not be discussed. Yet, the chapter will focus on various techniques that improve stability for a client that is interested in a file that is shared by peers listed in L. Obviously, there are current solutions to complete the download process as soon as L was created, yet the common techniques being used today present a download rate instability.

Examining file transfer analytically, we'd like to match the client with a set of sources that provide a stable and consistent rate of data. Since current P2P clients download portions of the file directly from the source peers that share the file, the stability of the download rate is bounded by the stability of the most stable sources in L. The source's stability might be degraded for many reasons such as packet loss, jitter, excessive delays or any other concurrently running software

that compete on the same bandwidth and CPU resources.

Another issue that might affect the download rate of the client is the availability of sources. A source might become unstable or unavailable and might increase the total time required to download the file. Existing file transfer systems limit the probability to find stable sources even more, as they manage fairness rules to prevent selfish free-riders in addition to using queueing techniques; thus, in such systems L is much smaller than its theoretical potential size of all peers that share the file. In systems that allow downloading from sources that received only portions of the file, as in eDonkey [21], the client might need to generate several requests until the missing file portions are found; this increases the total time required to download the file.

In systems where peers can serve packets also before completing the download of the file [50, 21], the excessive use of upload bandwidth will not only harm the download performance of the source peer, but will also have an effect on P2P uploads, since the peer will not be able to download new content at a satisfying rate. In addition, in some ISPs [33] P2P sources are subject to traffic shaping rules [63], which again degrade the throughput.

All of the above issues are typical to P2P based file transfer stem from the direct link between the potential source available upload rate and the user's behavior; this motivates the need for a new data transfer approach, adjusted to the characteristics of P2P networks and the behavior pattern of the end user.

2.10 Collabory Architecture

Feeding Networks Model:

A model is being presented here - where the file transfer rate for subscribers is higher and the consumed bandwidth is more stable by employing a software agent at the peers. Since a generic solution is being provided, it can be easily embedded in other works that present social groups or other bartering techniques such as tit-for-tat.

In our basic model, a peer that shares files (source) is called a supplier peer; a peer that requests files from supplier peers is called a consumer peer. Each consumer holds a list of

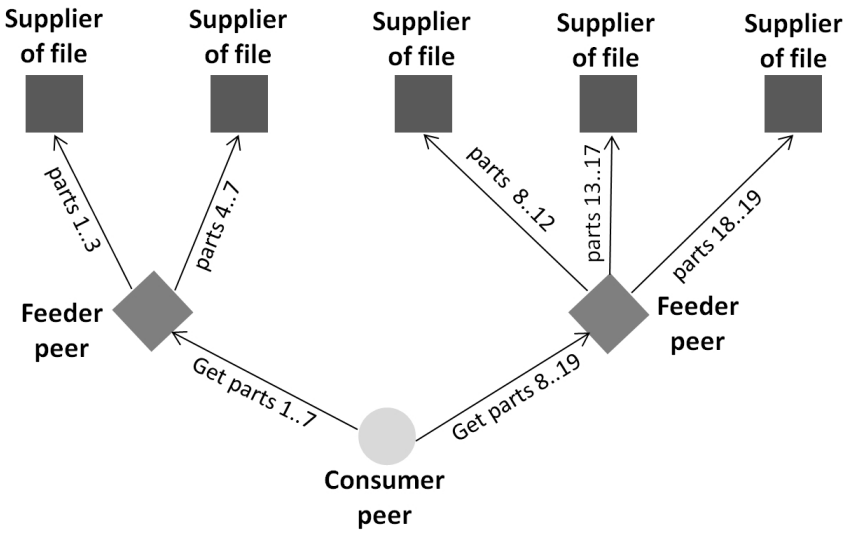

Figure 2.6: A one-level feeding structure

feeders, which is a group of peers that can download the requested packets from supplier peers or other feeder peers and transfer them to the consumer peer. In general, when a consumer peer wishes to download a file, it will ask each of his feeder peers to supply him with different parts of that file. Each feeder will open one or more connections with supplier peers that share the requested file (see Figure 2.6).

The list of feeder peers for each consumer can be built manually or automatically by tracing the stability metrics between the feeder and the consumer during past network activities. There are three major parameters used in order to analyze the compatibility of a potential feeder to a given consumer peer: availability, capability and infrastructure properties.

Availability: We'd like the potential feeder peers to be online and have limited network and CPU consumption when the consumer is about to start a new download process. Therefore, one would look for feeders that have a matching pattern of availability, meaning that they are likely to stay online and have low network and CPU consumption while the consumer is downloading. We'll use the term *fit* to address the above demands.

In order to find fitting feeders, Collabory log feeders' online periods (sessions) and the relevant network use and CPU utilization measurements within these sessions.

Let us term *Feedability* as the ability of a feeder to feed a consumer peer at a specific point in time i.e., the feeder is online and has low network use and CPU consumption.

Denote a Feedability function FA of feeder f, in session s at time t (time units after session initiation time) as:

$$FA_{f,s}(t) = \begin{cases} 1 & f_{s_{cpu}}(t) < Th_{cpu} \wedge f_{s_{bw}}(t) < Th_{bw} \\ 0 & otherwise \end{cases}$$

where $f_{s_{cpu}}(t)$ and $f_{s_{bw}}(t)$ are the measurements of cpu utilization and consumed upload bandwidth at after t time units from the beginning of session s (when the feeder went online). Th_{cpu} and Th_{bw} are the thresholds of cpu utilization and consumed bandwidth enabling the feeder to serve a consumer peer.

A potential feeder p is the most fitting feeder to a consumer peer (among all online feeders that have small RTT and low jitter with the consumer peer) if the average of its Feedability function $FA_{f,s}(t)$ over all of its sessions and a given time period starting now (when the consumer requested to start a new download) is maximized over all other feeders:

$$p = \arg\max_{f} \int_{t}^{t+k} \sum_{i=1}^{n_f} \frac{FA_{f,s}(t)}{n_f} \ ,$$

where n_f is the number of sessions that were logged by feeder f. k is being chosen as the length of a minimal time period for feeding before looking for alternative feeders.

Capability: One would prefer to choose feeders with lower bandwidth consumption and low CPU utilization as this gives us higher confidence in their ability to supply the client's network demands. Furthermore, the bandwidth provided by the feeder to its consumer is being limited in order to increase the immunity to unexpected bandwidth consuming events at the feeder.

For measuring these parameters, Collabory is being embedded as a special agent software that constantly logs the bandwidth consumed by different processes as well as the relative CPU utilization. The same agent can be used to alert consumer peers when an unpredicted resource consuming task was initiated - threatening the stability of the throughput between the feeder and the consumer peer. In this scenario, the consumer will replace the feeder with a different one.

Infrastructure: Network infrastructure aspects that affect the stability of a peer are not under the control of the user. Large delay variation (jitter), excessive packet delays and round trip time reduce the ability of TCP to ensure a stable connection. Packet loss might also contribute to the download rate instability especially between different regions. Our proposed agent software will log the round trip time and jitter parameters of each feeder it connects to. In time, it will construct a list of potential feeders that have the most stable download rate.

In addition, as some ISPs employ traffic shapers that [63] downgrade P2P download rate for external connections, by preferring feeder peers that reside in the same ISP as the consumer, it is possible to bypass the traffic policy and increase the throughput.

Analysis:

In Figure 2.7, $C_{regular}$ represents the case of normal file transfer - downloading from m sources, each supplying $\frac{MaxD}{m}bps$, where $MaxD$ is the maximum download rate of the client peer.

In $C_{feeder-based}$ however, the client downloads a file from m feeders, each of them dowloads from two sources: the 1st source supplies $\frac{MaxD}{m}bps$ and the second source supplies up to ϵ bps. The sources that supply ϵ can be used for short-term caching to ensure that the feeder peer can always supply $\frac{MaxD}{m}bps$ for its client.

In a working system, ϵ will dynamically change during the download process depending on the bandwidth supplied by the source peer. Following is the analysis of the simple model described above, comparing the Effective Download Rate (EDR) of each case, where p_x is the probability that a peer x (source or feeder) will deliver packets at full speed - the capacity of the modem:

$$
\begin{aligned}
&EDR(C_{regular}) = \\
&\quad m \cdot (\text{EUB of each } \textbf{source}) = \\
&\quad m \cdot p_s \cdot (\text{capacity of each source}) = \\
&\quad m \cdot p_s \cdot \frac{MaxD}{m} = p_s \cdot MaxD \ ,
\end{aligned}
$$

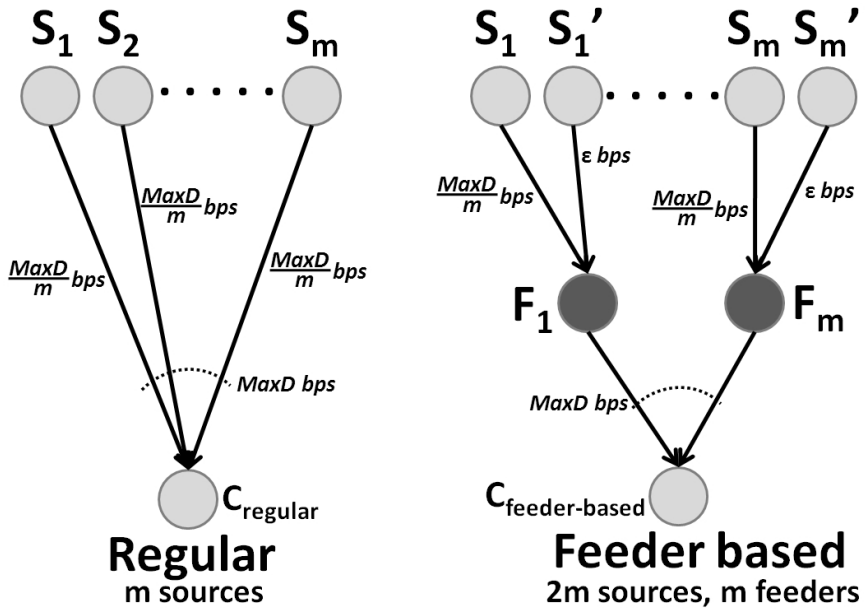

Figure 2.7: Schematic view of the general case

where EUB is the effective upload bandwidth.

$$
\begin{aligned}
&EDR(C_{feeder-based}) = \\
&\quad m \cdot (\text{EUB of each } \textbf{feeder}) = \\
&\quad m \cdot (p_f \cdot (\text{UB of feeder depended on its' sources})) = \\
&\quad m \cdot (p_f \min(\frac{MaxD}{m}, \sum \text{EUB (feeder's sources)})) = \\
&\quad \min(p_f MaxD, p_f p_s MaxD + m p_f p_s \epsilon) \ ,
\end{aligned}
$$

where UB is the upload bandwidth and EUB is the effective upload bandwidth. Thus:

$$
\begin{aligned}
&\frac{EDR(C_{feeder-based})}{EDR(C_{regular})} = \\
&\quad \frac{\min(p_f MaxD, p_f p_s MaxD + m p_f p_s \epsilon)}{p_s MaxD} = \\
&\quad \min(\frac{p_f}{p_s}, p_f + \frac{m p_f \epsilon}{MaxD}) \ ,
\end{aligned}
$$

meaning that $C_{feeder-based}$ will download at higher speed than $C_{regular}$ if $p_f + \frac{m p_f \epsilon}{MaxD} > 1 \Rightarrow \epsilon > \frac{(1-p_f) MaxD}{p_f m}$. Notice that as m grows, a smaller ϵ will satisfy the benefit of the feeder-based

solution. Likewise, if a bigger ϵ is allowed, less feeders may be used to gain the same results. This shows a great benefit of the feeder-based model over the regular model as it is possible to move the "risk" of a non-stable download bandwidth from the client to the feeder - that has potentially much more available download bandwidth than the client. Upon selecting stable feeders it is possible to reach better download stability while using even less stable sources, since the feeder has available download bandwidth that can be used for short-term caching. This means that a bigger ϵ is used to make sure that the feeder will be able to supply the requested bandwidth to the supplier. The asymmetric upload and download bandwidth does not affect our solution, since a feeder can theoretically download at full download speed to ensure the small upload bandwidth that it should supply the source.

Since ϵ is adjustable dynamically during the download phase, it affords using extremely weak and unstable sources from the P2P network and still not influence the stability of the download rate at the client, as long as the feeder manages to gather enough cache to be able to provide the requested rate by the consumer. Since it's possible to employ weak sources it is estimated that Collabory enhances existing networks' scalability as it increases the total number of potential sources because nowadays existing P2P applications tend to neglect weak sources.

For reinforcing feeders' redundancy, at the beginning of the download process an extra feeder is being initiated - that downloads packets from the end of the file and does not deliver them to the consumer until either the consumer already downloaded the rest of the file or in case a feeder has failed.

Scalability and Price: Collabory is an add-on that may be used in existing P2P networks. As Collabory utilizes weak sources that were not used by existing P2P networks, it increases the scalability of P2P networks considering that the same quality of service (average download speed) of existing networks is maintained since it increases the supply of available sources. There is a trade-off between using the extra available resources for scalability (more clients using the new weak sources) or for QoS (more weak sources for existing clients) and Collabory's architecture allows controlling this trade-off to reach the desired characteristics. As current P2P networks' scalability is bounded by anti-free-riding mechanisms, the utilization of unused resources such as

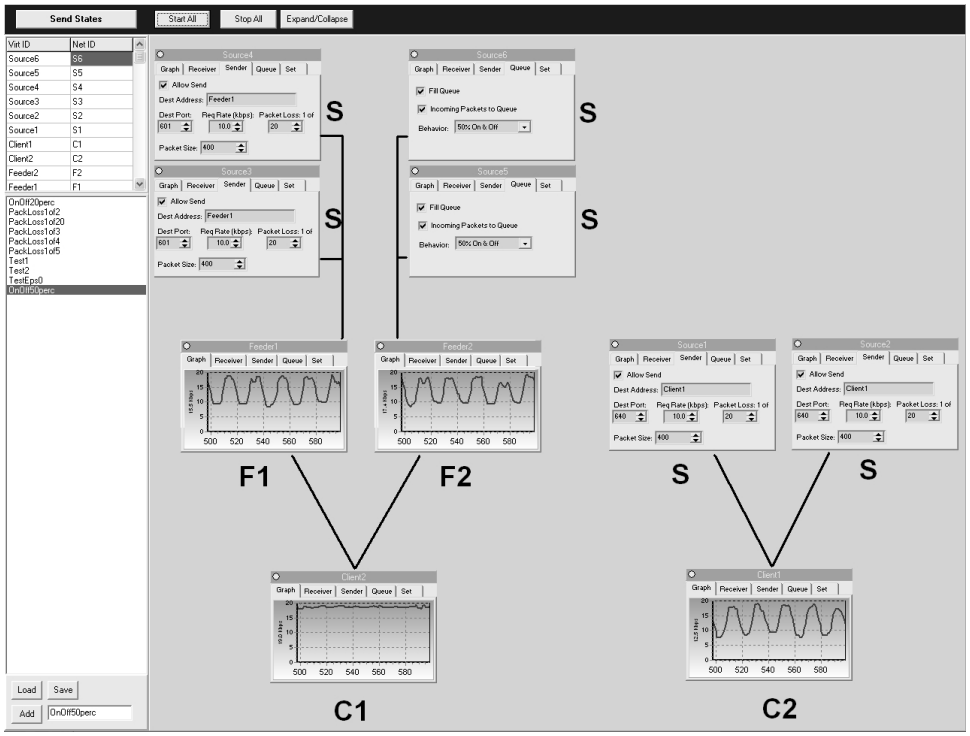

Figure 2.8: Collabory test suite

weak sources by Collabory allows reducing the excessive usage of fairness rules.

Compared to the regular case, using collabory does not have to exploit more sources (that share the requested file) and in the case the number of sources are increased, still the total number of total bytes sent from the sources to the client is the same. Yet, the extra use of Feeders increase the total traffic of the network - but to sweeten a bitter pill - Feeders may be localized (thus limiting the burden on ISPs) if the Feeders are being chosen from nodes in the vicinity (same ISP) of the client.

2.11 Testing Collabory

Test Suite:

For conducting the tests, a special test suite was built that allows defining structures of connections between 3 types of nodes: source peers, feeders and clients. The suite is composed of a

monitoring utility and a set of agents that can emulate the behavior of any of the 3 types of nodes. The agents may run in different machines and submit their performance metrics to the monitoring utility.

As can be seen in Figure 2.8 the test suite is configured to compare the feeder based model with the regular one.The client/consumer peer $C1$ is connected to 2 feeders $F1$ and $F2$ and each feeder is connected to 2 sources, marked with S. The client $C2$ represents the regular case thus it is only connected to 2 sources. The graphs attached to each feeder and client peers shows the download rate measured at that peer. In this specific experiment burst scenarios were examined; i.e., all source peers were set to behave in a repeating pattern of sending at 50% of their maximal upload bandwidth for 10 seconds followed by additional 10 seconds of sending at full speed. It can be noticed that the download rate at the feeders and at $C2$ behave the same, due to using sources with the same behavior. However the client that downloads from the feeders $C1$ gained a stable and higher download rate.

Measurements:

The tests were conducted in two different settings. The first one set the maximum download throughput of peers to 920Kb/Sec - common to broadband. Upload is limited to 460Kb/Sec. Packet loss is set to 1 to 20 for all source peers. By employing the behavior of 50% mentioned above the client received 866Kb/Sec at $C1$ (feeder based) and 648Kb/Sec at $C2$ (regular). Notice Figure 2.9 for the results of 80%, 50% and 0% (full transmission for 10 seconds on and off) where the extreme behavioral settings of the sources had only slight effect on the download rate of $C1$ compared to a major decrease in $C2$. The rest of the results are shown for the setting where the maximum download throughput of all peers is set to 20Kb/Sec and the upload is bounded by 10Kb/Sec. This was chosen to show the benefit of Collabory on extremely weak peers that are hardly being used in existing networks because of their unstable nature and low bandwidth.

Figure 2.10 examine different values of ϵ to see how it affects the performance of feeders. All sources in the system are set to the 80% settings of the previous test, including the sources that transmit ϵ which means that they will repeatedly transmit 0.8 ϵKb/Sec for 10 seconds and then

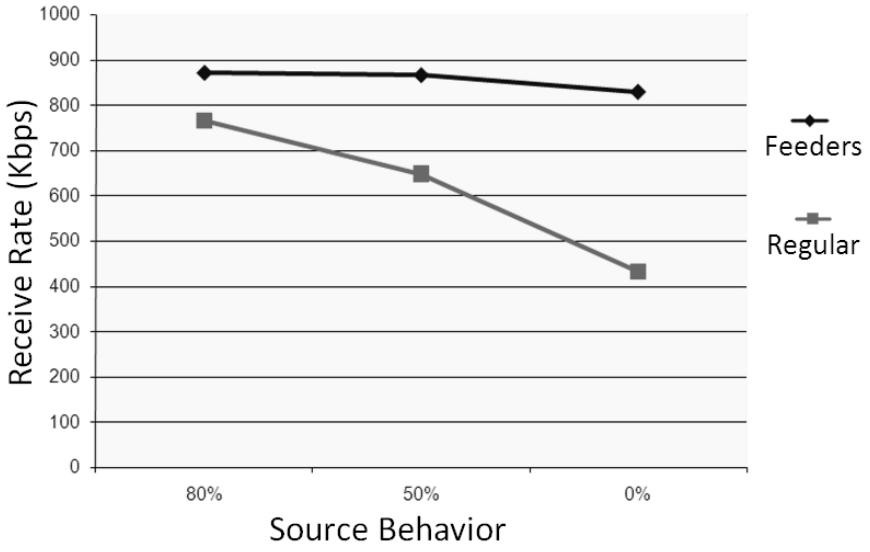

Figure 2.9: Varying source behavior

ϵKb/Sec for the following 10 seconds. Given larger values of ϵ allows the feeders to hold a cache for a longer period of time and this way be able to transmit the cache content to $C1$ accordingly. Notice that when ϵ is set to 2.2 the cache content was increasing consistently thus allows the feeder to transmit $C1$ as if it was a stable source. In this scenario $C1$ received stable download rate of 18.9Kb/Sec.

The case of using weak source peers for the feeder was also tested (Figure 2.11). For the regular method two sources of 10Kb/Sec were set with the behavior of 80% mentioned above. For the feeder method the following different test settings was set - A: 4 sources of 6.0Kb/Sec under 80% behavior. B: 8 sources of 3.0Kb/Sec under 80% behavior. C: 8 sources of 4.0Kb/Sec under 50% behavior. In all of the tests a stable increased rate was achieved in the feeder case compared to unstable rate in the regular case.

eMule/eDonkey Hooking Experiment:

The solution was implemented on a set of 3 machines connected to ADSL broadband connection through different ISPs using 1.5 Mbps ADSL links. Two computers served as feeders for the third machine. Emule was hooked by providing it a set of file portions and generating on-demand maps of the portions planned to be downloaded (*.part.met* files), allowing to "order" any piece

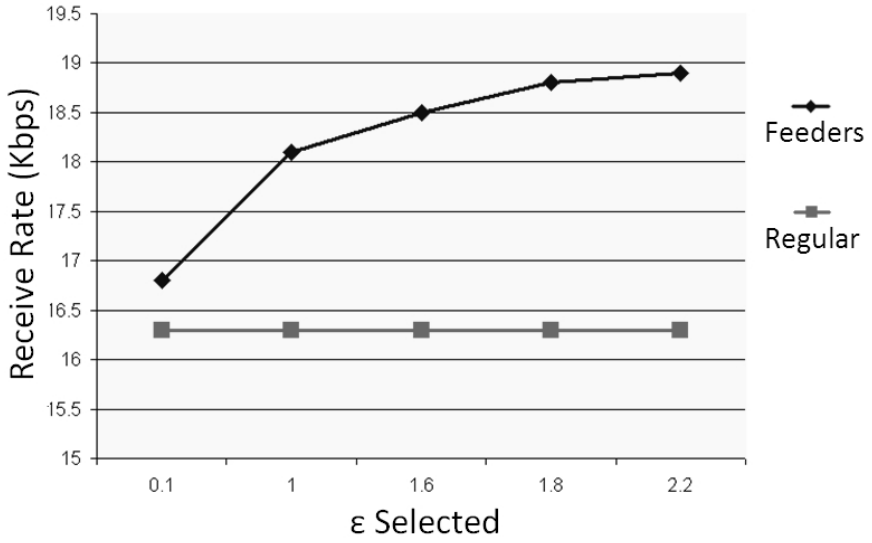

Figure 2.10: Different ϵ values

of file wanted at any period of time. Each feeder managed multiple copies of eMule, and opened multiple requests for specific file portions on demand.

The experiment ran for 10 periods of 4 hours at different parts of the day covering peak hours as well. Collabory constantly logged the download rate of the client. Every 30 minutes, the experiment was switched between Collabory and the normal eMule client. As shown in Figure 2.12, an average of 1.41 Mbps for the Collabory supported client was measured and an average of 0.92 Mbps for the normal eMule client.

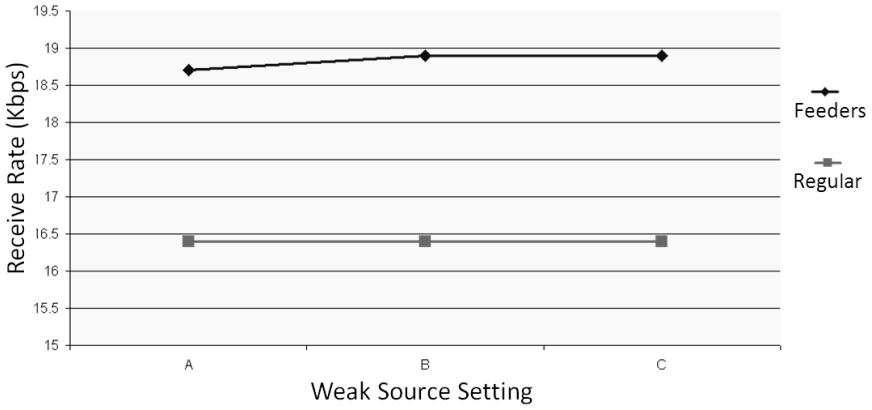

Figure 2.11: Different settings of weak sources

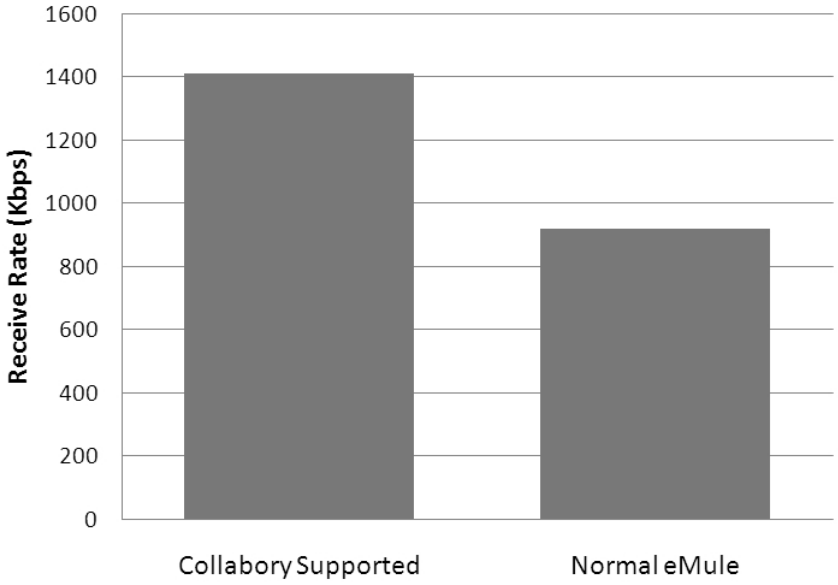

Figure 2.12: Collabory on eMule

Chapter 3

User Behavioral Patterns

3.1 Predicting User Behavior

Emerging large scale file sharing Internet applications base their infrastructure on P2P technology. Yet, as presented in the previous chapter, the characteristic fluctuational throughput of source peers affect the QOS of such applications which might be reflected by a reduced download rate in file sharing or even worse - annoying freezes in a streaming service. A significant factor for the unstable supply of source peers is the behavior of the user that controls other processes running on the source peer that consume bandwidth resources.

If one can learn the behavioral patterns of the user that runs various applications that consume bandwidth, it will be possible to utilize this information for building better P2P networks. In this chapter Collabrium is presented - a collaborative solution that employs a machine learning approach to actively predict the load in the uplink of source peers and alert their clients to replace their source. Experiments on home machines demonstrated successful predictions of upcoming loads and Collabrium learned the behavior of popular heavy bandwidth consuming protocols such as eMule & BitTorrent correctly with no prior knowledge.

Yet, the instability of throughput in P2P networks is not limited only to file sharing applications; In theory, peer-to-peer (P2P) based streaming designs and simulations provide a promising alternative to server-based streaming systems both in cost and scalability. In practice however,

implementations of P2P based IPTV and VOD failed to provide a satisfying QoS as the characteristic fluctuational throughput of a peer's uplink leads to frequent annoying hiccups, substantial delays and latency for those who download from it. Here again, the user's selection of applications that require bandwidth, play a significant factor for the unstable throughput of peers' uplink with processes running on the source peer that consume bandwidth resources.

Following Collabrium, Maxtream is presented - an adaptation of our machine learning based solution for streaming peers, coordinating source peers exchanges between peers that suffer from buffer underrun and peers that enjoy satisfactory buffer size for coping with future problems.

3.2 Collabrium Introduction

P2P networks have received significant attention from academia and industry in the past few years for various reasons [95, 90, 107, 87]. While anonymity and copyright infringement played a substantial role in the expansion of these networks over the early years of P2P, business oriented factors such as cost of delivery and maintenance are reflected on recent initiatives to establish P2P based large scale Content Delivery Networks (CDNs), file transfer and streaming services such as IPTV and VOD. Accordingly, the key parameters in designing P2P networks have changed. While in the past it was legitimate to embed mechanisms like queues, anti free-riding and fairness [74] rules at the expense of QOS issues like download speed and service continuity [54, 42, 26, 104, 75, 43, 48], business considerations dictate a different approach where QOS can no longer be compromised.

Unfortunately, CDN services that are based on the popular BitTorrent protocol, suffer from an unstable download rate and hardly exploit the available download capacity at the consumer side [44, 37]. One design [84] aim to solve this issue by viewing content distribution as a global optimization problem using central Coordinator servers.

Recent studies have shown that the major factor influencing QOS in P2P networks is the behavior of users at the source peers [56,88] - taking occasional actions that heavily use bandwidth such as sending Email, online games and running other P2P applications in parallel. This behavior

leads to fluctuational rate of packets for the client peer which might be reflected by a reduced download rate in file sharing networks or latencies, delays and freezes in streaming P2P networks.

Collabory contributed the notion of *Feeders* - normal peers that are currently online and free to serve as a proxy cache for the benefit of a client peer that wishes to download a file. The Feeder stores the file's pieces from several unstable sources and offer the pieces to the client in a stable fashion. In order to guarantee the stability, given client were matched with potential feeders that have good connectivity with the client like minimal packet loss, small delay, low jitter and are likely to stay online while the client is downloading. In order to guarantee the long service of a suitable feeder, historical statistics of overlapping online time periods between the client and the feeder was used. Yet, this strategy misses many potential feeders and sources that have good quality connection with the client but weren't selected since the overlapping online time periods were not long enough to provide confidence that the feeder won't disconnect while the client is downloading from it. If it was possible to predict that a potential feeder's uplink is about to be dropped, our solution could alert the client to select an alternative feeder prior to that drop. This will significantly increase the amount of potential feeders as the solution will no longer be restricted to bounds dictated by historical statistics of overlapping time periods.

Follwing Collabrium is being presented - a collaborative software agent based on a machine learning approach, that employs Support Vector Machines (SVM) [99] to actively predict load in the upload link of source/feeder peers and accordingly alert the client to select alternative source/feeder peers. Collabrium discerns patterns of communications with no prior knowledge about any protocol which allows it to predict new protocols as well. The solution is reinforced with an optional agent that monitors process executions and file system events, that improves the prediction even more. Experiments demonstrate high accuracy in predicting uplink load on popular P2P protocols such as eMule and BitTorrent, and ability to handle encrypted protocols such as obfuscated eMule as well.

3.3 Collabrium Related Work

Some research papers proposed to embed machine learning algorithms in P2P networks. [54] uses Maximum Likelihood Estimation to evaluate trust and reputation for peers bilateral interactions. The authors in [42] employ SVM for selecting neighbors in an unstructured network as Gnutella, to optimize the path of queries in the network. In [26] an SVM based classification tool for ISPs is presented to differentiate between P2P-TV protocols by counting packets in a given time frame.Both [104] and [75] classify classes of protocols according to predefined templates of sequences of IP addresses and port values for recognizing anomalies at the backbone.The result in [43] predicts latency between nodes based on IP address values. An SVM based solution is proposed in [48] for predicting the available bandwidth in an internet path by transmitting probing packets. These papers do not offer a method to predict load in the uplink of source peers actively, and do not address the influence of user behavior on the source's available bandwidth.

3.4 Problem: To P2P Or Not?

Recent measurements of broadband usage patterns in ISPs reveal a surprising rising trend that should concern the P2P research community: new server based services are growing in traffic at the expense of P2P traffic. While P2P is still responsible for more than 60% [30]of all upstream data in ISPs, it is claimed [7] that subscribers are increasingly turning to alternatives such as File Hosting web sites like RapidShare and MegaUpload or Media Content web sites like NetFlix [27], since they enable much faster download speed compared to P2P networks. RapidShare was already ranked as the 12^{th} web site in global traffic rankings according to Alexa.Com web traffic rating. Another study [77] approves the above and claims that web sites like RapidShare grow in traffic on the expense of P2P. BitTorrent for example - the most popular P2P protocol, suffers from unstable download rates and hardly exploits the available download capacity [44,37]. Netflix with 22% of traffic has overtaken BitTorrent (21.6%) as the largest component of Internet traffic in fixed access networks in North America [30].

The above problems put P2P technologies in question for commercial system designers, due

to QOS problems. As most P2P systems already run a best effort approach by prioritizing peers with minimized infrastructure problems like delay and packet loss, they still miss a key factor in degrading P2P performance - the user behavior. In addition, this approach is blind to a large number of weak sources that remain unused, while the small group of strong sources are exploited and overused [73].

Collabory [68] analyzed the factors for the instability of source peers in P2P networks and found that the aspect that has the greatest impact is the behavior of users at source peers. The most obvious occurrence is the case where the user at the source peer invokes applications that heavily use bandwidth such as Email clients, online games or other P2P applications. By doing so, the bandwidth available for the client connected to that machine may be drastically reduced and become significantly unstable. Recent studies confirm that the major factor that has direct impact on QOS in P2P networks is the behavior of users at the source peers [56, 88]. This behavior leads to fluctuational rate of packets for the client peer which might be reflected by a reduced download rate in file sharing networks or latencies, delays, hiccups and freezes in streaming P2P networks.

3.5 Collabrium Architecture

3.5.1 Behavior Aware Feeders

As presented in Collabory [68], the key concept of Feeders is that normal peers that are currently online and free to serve can be used as a proxy cache for the benefit of a client peer that wishes to download a file. The Feeder stores the file's pieces from several unstable sources and offer the pieces to the client in a stable fashion. In order to guarantee the stability, a given client is matched with potential feeders that:

1. Have good connectivity with the client. i.e. minimal packet loss, small delay, low jitter;
2. Are likely to stay online as long as possible while the client is downloading.

For ensuring connectivity quality, historical measurements can be used or evaluating locality according to the IP address. For ensuring that a feeder will stay online statistics of the periods where the feeder was online is gathered and maintained a certain threshold of available upload bandwidth; then the client is matched with the feeder that is likely to stay for the longest period of time.

Yet, the strategy of limiting potential feeders to those that are likely to stay online for long periods has several weaknesses:

1. Some feeders might be missed if they have good connectivity with the client, are currently online but statistically the probability that they will stay online is low or in the case that repeating online periods couldn't be realized for these feeders due to lack of history or a varying behavior;
2. Since maximal online periods are being looked for, it might be possible to miss many potential feeders that could serve for significantly shorter periods;
3. While it's possible to guarantee an effective download rate for our client for long periods, it's impossible to guarantee that the feeder won't have drops in it's upload for short periods during the download process.

The first 2 weaknesses substantially limit the scalability of the original proposal. The 3^{rd} one limits it only to file sharing applications, while the user behavior factor on P2P streaming networks remained unsolved. While the structure of integrating feeders proved to be highly effective in stabilizing and maximizing throughput [68], in order to make it much more scalable as well as optimize it to streaming networks, a new approach which sharpens the focus on the real "problem maker" of instability - the user is offered.

Collabrium let **any** peer that has potentially good connectivity with the client to serve as a feeder. Yet, instead of looking at historical statistics of online periods, it employs an active approach that is based on machine learning to predict when a feeder is about to downgrade the stable throughput it provides to the client. As in Collabory, Collabrium employs a software agent that resides in peer computers, to allow the prediction algorithms to monitor the system activity.

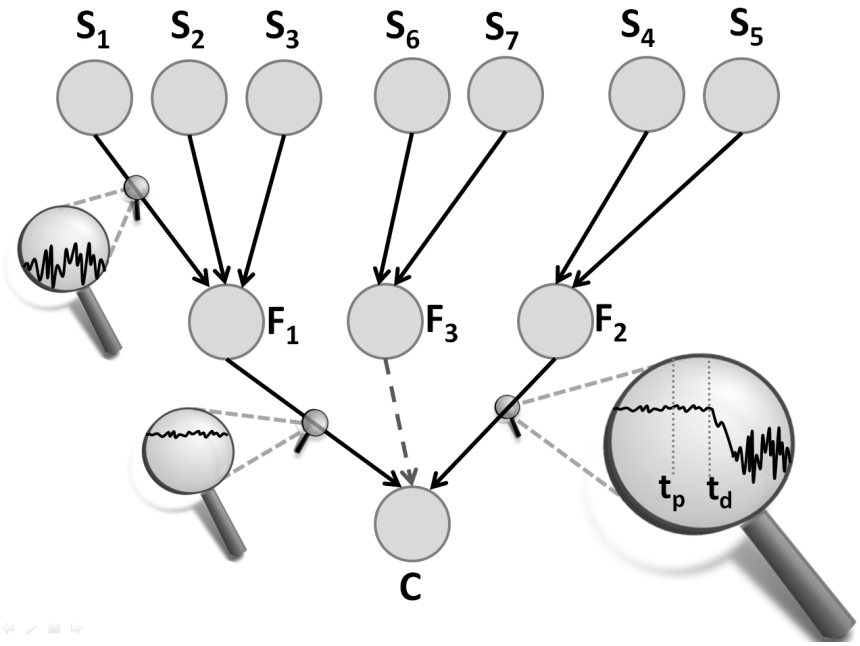

Figure 3.1: Collabrium concept chart

Figure 3.1 illustrates the concept of user behavior aware feeders. C represents the client that downloads a file or a streamed media content. S_i represents the sources in a regular P2P network and F_1,F_2,F_3 represents the feeders. Notice that the throughput between S_1 and F_1 is low and unstable, and assume the same for all of the connections between sources and feeders. Yet, the throughput between F_1 and C is high and stable, as mentioned above that feeders are selected as peers with good connectivity with the client. Now let's assume that that C begins using F_1 and F_2. F_1 has enough available upload bandwidth to supply C a stable throughput. As for F_2, notice that at the beginning it provided stable throughput to C as well, but at time $t_p - \epsilon$ the user at F_2 opened another P2P software or any other process that consume upload bandwidth. A few seconds later, at time t_d, the throughput between F_2 and C dropped and became unstable due to the new software/process. Collabrium's agent that runs on F_2 predicts at time t_p that it will soon have to share it's upload bandwidth with another process, therefore it immediately notifies C to replace a feeder. C connects to F_3 and by t_d, C no longer communicates with F_2, thus C didn't experience any drop in its download rate.

Collabrium can be implemented over any P2P existing protocol as the sources in Figure 3.1 can be sources of any P2P network and Collabrium don't manage them, but only request for file portions.

3.5.2 Monitoring Module

The monitoring module is responsible for collecting data for the learning module. It acts as a packet sniffer for both inbound and outbound links and logs packet arrival time, header and payload. Though the network collected data alone was enough to provide sufficient prediction accuracy, additional data was logged for file system activity and active process list as in some cases it can further improve the prediction. The file system information is logged by a Win32 IFS (Installable File System) hook - a DLL that monitors file system events such as read, seek, write etc. While the monitoring is done as a background process, the information in a database is being logged only for a limited time - while the learning process is active. This time should be sufficient to gain enough information so that the user behavior can be predicted in the future, given a set of measurements. For the average user, our experience showed that logging along one full day is enough. It's recommended to re-run the learning process from time to time, in order to adapt to the user's new habits and trends.

3.5.3 Learning and Prediction Modules

The learning module extracts the data that was collected by the monitoring module into sets of features and values for the learning algorithm. The core of this module is based on a Support Vector Machines (SVM) [99] classification algorithm, yet the assembly of *feature:value* pairs is not straightforward as elaborated here.

The learning algorithm should be able to link the collected data to the occurrences of traffic load in the uplink. As illustrated in Figure 3.2: S_1,S_2 and S_3 are sessions. A session is identified by source IP and port, and destination IP and port, thus it begins with the first packet that was sent between our peer i on port x and a peer j on port y and ends with the last message

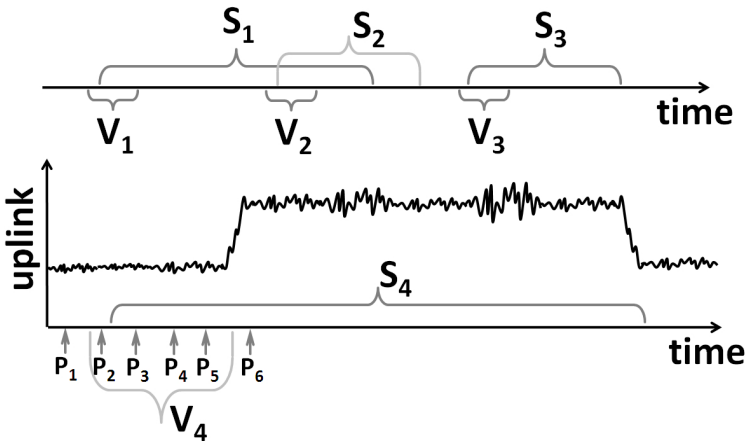

Figure 3.2: Load Vicinity Pattern Prediction

that was sent between the same peers on the same ports. If the time between 2 sequential messages is larger than a specific predefined threshold, it is sees it as 2 sessions (aging). Notice that sessions might overlap as in sessions S_1 and S_2 but still it's possible to identify the session of a packet using the key of IPs and ports. V_1, V_2 and V_3 are the vicinities of S_1,S_2 and S_3 respectively. A vicinity is a collection of packets that were collected around a predefined time period at the beginning of each session. Notice that the vicinity begins a few milliseconds before the beginning of a session. In session S_4 and its vicinity V_4 the change in uplink utilization due to that session is presented. Notice that typically, the load in the uplink begins a few seconds after the beginning of a session and not immediately, as in most P2P algorithms the very first messages are used for preliminary negotiation, thus the packet P_3 and its neighbors can be used to predict the upcoming load and still have enough time to notify the client about it. In some protocols, packets that are in the vicinity but precede the session like P_2 can tell us about the upcoming load due to some negotiation between the peers or between a peer to its supernode. Collabrium's key strategy is that it predicts a traffic load by examining the properties of packets that precede the load - meaning the packets in the vicinity of sessions that loaded the uplink. Following, different properties that proved to be significant for prediction are presented along with their extraction techniques:

Load Vicinity Pattern Prediction

In this method the first bytes (15 bytes were found to be effective) of the payload of each packet that is in the vicinity are being looked at and *feature:value* pairs are extracted for SVM so it can learn specific patterns. For example, in eMule's client-client protocol, the 1^{st} byte is always *0xE3* and in the handshake message the 6^{th} is always *0x01*; They are marked as *byte:value* pairs that form a pattern: *1:0xE3*, *6:0x01*. SVM is targeted to realize these patterns out of the messages in the vicinity. Since close values such as *1:0xE3* and *1:0xE4* might belong to completely different protocols or different messages of the same protocol, SVM can't be introduced to these values directly as it will not relate them as discrete values. Therefore, the most popular *byte:value* instances of packets in the vicinities of all sessions are collected while giving priority to *byte:value* pairs that appear in different sessions.

Notice the algorithm pseudocode in Figure 3.3. First, run over all packets in the log of captured network activity - *PacketLog* over a predefined time frame (24 hours in our experiments). Gather packets with the same source IP and port and destination IP and port into a session and store the accumulated byte count of packets' payload in a session list - *SessionList*. Short sessions are deleted from the list as they do not indicate a significant traffic that might threat the available throughput in the uplink. Then, look for packets that reside in the vicinity of the beginning of long sessions. In the algorithm it appears that it looks only for one session for simplicity, yet the experiments addressed all sessions that begin in the vicinity of a packet, as there might be more than one. As can be seen in Figure 3.2, packets that precede the beginning of a long session are also interesting as they might indicate of an upcoming session. Collabrium creates *ByteValueList* - a list that stores the counts of specific byte values in specific index of location in the packet's payload. This will assist us in collecting the most popular *Byte:Value* instances that take part in a pattern that identifies one or more packets that precede traffic load. Notice that in case of multiple instances of the same *Byte:Value* pair, only one instance is being related, as the interest is in patterns that reside in as many sessions as possible and minimize the influence of a pattern that might have repeated itself in a specific session.

Finally, the training set is supplied for SVM; Each item in the training set contains the following

```
For each packet p in PacketLog do
   SessionList[p.SrcIP,p.SrcPort,p.DstIP,p.DstPort].ByteCount += p.ByteCount
For each session s in SessionsList do
   If s.ByteCount < BYTE_COUNT_THRESHOLD then
      SessionsList.Delete(s)
For each packet p in PacketLog do
{
  s = SessionsList.GetNearestSession(p.TimeStamp)
  If SecondsBetween(s.StartTime, p.TimeStamp) < VICINITY_THRESHOLD then
     For i = 1 .. MAX_PATTERN_SIZE do
        If Not ( s.ID in ByteValueList[ i, p[i].Value].Sessions ) then
        {
           ByteValueList[ i, p[i].Value].Count += 1
           ByteValueList[ i, p[i].Value].Sessions.Add( s.ID)
        }
}
ByteValueList.SortByCount
```

Figure 3.3: Extracting popular *byte:value* pairs

features: Source IP, Source port, Destination IP and Destination port. Then, a feature per each of the top popular items in *ByteValueList* is created; i.e. if the most popular *byte:value* pair is *5:0xE3* and the value of the 5^{th} byte of the packet is *0xE3* then insert *1:1* as a *feature:value* pair for the training item; if the second most popular *byte:value* pair is *3:0xB6* and the value of the 3^{rd} byte of the packet is *0xC2* then insert *2:0* in the training set since the values are different and so forth for the next popular *byte:value* items, up to a certain amount of features (the top 100 popular were found to yield satisfactory results). Label as *+1* training items that represent packets in the vicinity that contain at least one instance of the top popular *byte:value* pairs. The training set are also supplied with packets that are not in the vicinity and label them as *-1*. When the prediction module is running to look for upcoming loads in the uplink, it will simply propose recent captured packets' properties to SVM with the appropriate features and SVM classifies the packet as leading to uplink load or not. For differentiating between positive and negative labels, SVM is fed with additional randomly selected packets (not in the vicinity of the beginning of sessions) and label them as negative.

Packet Size Sequence Prediction

While looking at the data that was captured in the beginning of sessions, interesting phenomenon in P2P protocols was noticed - the byte count of the first packets form a sequence that repeats itself with minor differences for nearly all sessions of the same protocol. For example, a typical packet size sequence for eMule is $\{0, 0, 0, 125, 108, 11, 11, 41, 83, 77, 55, 55, 22\}$. Since some slight differences in the sequence were noticed, it's not possible to use it as a serial set of features for SVM as in some cases the value of *108* in eMule might appear as the byte count of the 5^{th} packet while in other cases it will be the byte count of the 6^{th} packet due to an extra packet. Therefore, these values are related as a histogram, and simply define a predefined number of features (30 was found to yield good results) for the most popular byte count values in a similar manner to the previous algorithm. For example, if the most popular byte count is *125*, the training set is supplied with a feature with a value of 1 if the vicinity of the examined packet contains at least one packet with this byte count or 0 if not. This technique is also applicable for obfuscated/encrypted protocols, as it does not rely on the packet's payload.

File System and Process List Prediction

In a few cases it is not possible to predict uplink load just by looking at captured packets. For these cases Collabrium monitors file system and process execution events and let SVM link between those events and the load. The method is based again on the same techniques already used: in the vicinity of the beginning of sessions that raised the load on the uplink, look for the most popular folders accessed (that are not system folders) and label a positive item in the training set with features that test if there was access to a specific popular folder or not. The same is done for the list of processes that joined in the time frame of the vicinity.

Prediction Module

In the prediction module, while packets are being captured, the properties mentioned above are extracted and served to the SVM algorithm. In case that SVM classified the packet as leading for load and the uplink used bandwidth is larger than a predefined threshold, Collabrium notifies

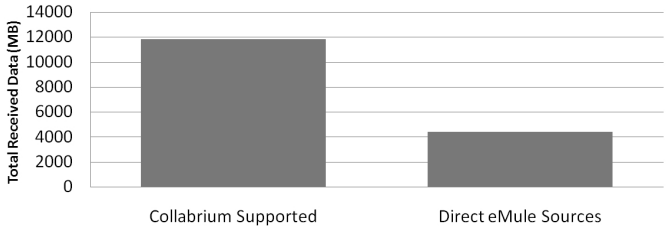

Figure 3.4: Received data during test periods

the client to select an alternative feeder. The same is done for file system events and processes.

Motivation and Incentives

The clear benefit of using Collabrium is that you receive better QOS when you need it in response to your assistance for maintaining the QOS of others. The active prediction approach allows almost any peer to serve as a feeder when needed thus the system is highly scalable. Moreover, in order to enjoy Collabrium's benefits it is not a must to install it in all peers as it can be used by a closed-group of peers that are part of a social network where the more friends you make, the better stability you can guarantee.

3.6 Testing Collabrium

3.6.1 SVM settings

The tests were conducted using SVM-Light library. The Radial Basis Function (RBF) Kernel performed best for the experiments. RBF kernel - $K(x, y) = e^{-\gamma\|x-y\|^2}$ can map samples to higher dimensional space, thus unlike the linear kernel it can handle the case where the relationship between class labels and attributes is nonlinear. After experiencing with different γ and C values for various scenarios, a γ of 0.5 and C of 2 was found to provide satisfying results. The price of running SVM in a home computer is not significant as the training phase can be run in time frames when the CPU is not busy. As for the prediction phase, CPU utilization was under 10%

while the prediction was running.

3.6.2 Implementation

The solution was implemented on six machines A,B,C,D,E and F - all were connected to the Internet using separate broadband connections of $1.5Mbps$ download and $0.5Mbps$ upload. All were used regularly by home users during a test period of 6 days. The prediction module was installed on machines B,C,D,E and F that were used as feeders for A. Prior to the week of tests, the learning module ran for 24 hours on each machine. Along the week of tests, A downloaded from eMule on 10 randomly selected time periods per each day. Per each period, A downloaded directly from eMule sources for 20 minutes and then downloaded indirectly using the feeders for 20 minutes. The same set of 5 files were used for all tests and after each test the received files were deleted to prevent caching for the next test. On the tests that used feeders, 3 feeders were selected randomly and a feeder changed only when the prediction module of this feeder informed about an upcoming load in its upstream. The users at B,C,D,E and F were using different applications that typically consume upload bandwidth from time to time such as eMule (also obfuscated), BitTorrent, LimeWire, Email clients and online games.

Figure 3.4 illustrates the total received data during the test periods. While the Collabrium solution provided 11820 MB of files from eMule for A, the regular - direct eMule sources solution only received 4440 MB.

Figure 3.5 examines various protocols that use the upstream and Collabrium's ability to predict an upcoming load per each protocol. 5 hours of activity were captured on each of these protocols separately. Then mapped all large sessions (more than $1MB$) and counted the cases where Collabrium predicted a large session successfully. Notice that in the fourth case, all protocols were running on the same machine for 5 hours, to examine the case where the vicinity contains messages of multiple protocols. The predictions were made for two settings - network only that relied on packets log and a combined prediction that was based on network, file system and process list as mentioned earlier. Notice that the benefit of file system activity and process list was minor and encouraging results were received with network-only prediction. Along all tests

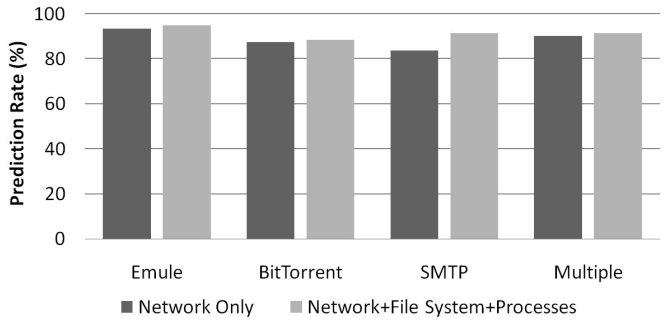

Figure 3.5: Prediction rate for different settings

the time between the prediction and the beginning of load in upstream was measured and found that this client has between 3 and 6 seconds to replace a feeder in order to prevent reduction in throughput.

Figure 3.6 visualizes experimenting different vicinity sizes and measuring the appropriate prediction success rate. The leading part of the vicinity ($3^r d$ of its size) is placed before the beginning of a session - to allow prediction using packets that might lead to a session (like an interaction between a peer and a supernode prior to the file transfer between peers). Notice that small vicinities of between 1 and 2 seconds do not consist enough information to predict an upcoming load with high success rate. Vicinities larger than 4 seconds tend to create more noise than useful information for prediction and accordingly the prediction success rate degrades. In the experiments only minor false positive predictions of up to 2% of the total predictions were received.

3.7 P2P Streaming

Following the increased popularity of P2P file sharing applications and a substantial growth in the creation of user generated content (UGC), came a corresponding increase in initiatives for building the ultimate streaming media network with minimal cost and high quality, both for broadcasting live TV and video on demand applications. Yet, so far there is no known winning recipe for building such networks. While there are some implementations that proved to be scalable and

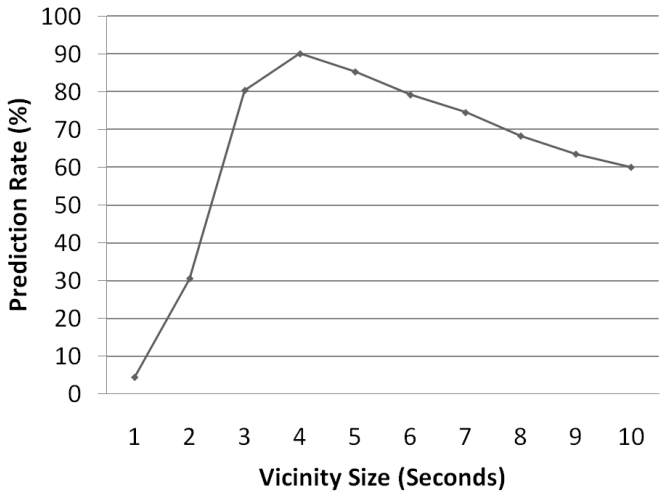

Figure 3.6: Prediction rate per vicinity size

cost effective, they failed with the end user's experience due to a mixture of low quality video content and QoS problems.

Popular implementations of P2P based networks in the industry such as Joost [24] and PPLive [29] were reported to often suffer from a variety of QoS problems [100, 13, 15, 5] like broken streams, streaming audio/video hiccups, substantial latency and major delays in live broadcasts. While in server based streaming services it is possible to solve QoS problems with buffering/caching, the instability of peers upload in P2P streaming networks require a much larger cache, which makes it impractical - as even when networks' policy allowed extremely unbearable latencies of up to 2 minutes [100, 65], users still faced the above problems. Instead of watching movies using streaming services, users still prefer to wait for the media content to be downloaded using either file sharing services or follow the latest fashion by using alternative non-P2P services such as RapidShare, MegaUpload and NetFlix [7, 77, 27] for downloading movies. Recently it was shown that Netflix with 22% of traffic has overtaken BitTorrent (21.6%) as the largest component of Internet traffic in fixed access networks in North America [30].

Studies have shown that the major factor that has direct impact on QoS in P2P networks is the behavior of users at the source peers [56, 88] - taking occasional actions that heavily use

bandwidth such as sending Email, online games, running other P2P applications in parallel or even terminating the process of the P2P network while it is streaming.

While it is reasonable to claim that users that watch online TV will not run other processes in parallel, in reality many users tend to keep their file sharing P2P applications and Email clients running in the background. In addition, for many other users watching online broadcasts such as sports events (in a small frame, not in full screen) is a background task while they use other software. This behavior leads to fluctuational rate of packets for the client peer which might be reflected by latencies, delays and hiccups in P2P streaming networks.

Thus there is a need for a solution that will address the problem of user behavior in P2P streaming networks and provide a mechanism that is able to absorb the instability of source peers and allow stable throughput at the client side.

Following, Maxtream is being proposed - a variation of Collabrium for streaming networks, that employs Support Vector Machines (SVM) [99] to actively predict load in the uplink of streaming peers and coordinates source peers exchanges between peers that suffer from buffer underrun and peers that enjoy satisfactory buffer size for coping with future problems.

3.8 Maxtream Related Work

Previous research designs proposed several ideas to improve P2P streaming networks' performance. [97] presented a topology of clusters and a multicast tree is built on top of hierarchy of clusters. [56] presented a simulated P2P VOD network based on maintaining an application multicast tree. [53] proposed tree based overlay intended for hundred of clients in a multicast group. [105] offered a P2P VOD network that pre-caches content on peers. [62] presented a P2P VoD that streams the video between the clients in a tree. [47] is a high-bandwidth content streaming/distribution system that is built upon Pastry and Scribe. [83] employed distributed streaming to obtain the content from multiple peers simultaneously. [88] progressively evaluated various combinations of senders to determine a subset of the senders that can collectively provide maximum throughput. In [98] the authors present a scalable auditing-based technique for en-

forcing fairness in a live-streaming system; the solution depends on a hybrid design that employs a set of trusted global auditors.

Most of the above designs do not relate to the user behavior influence on the streaming process. Those that do mention behavior, limit it only to node leaves. Most research papers tend to "divide the world" into strong peers that can take part in the streaming network and weak peers that can't. The decision if a peer is strong or weak is usually made preliminary based on measurements such as ping roundtrip time. Once it was decided, the decision remains unchanged and a peer that was strong cannot turn into a weak peer anymore. However, in reality peers often change the throughput they supply to their clients due to other processes that consume upload bandwidth and not only while leaving the network. A strong peer might become weak at some stage and a weak peer might become strong. None of the above works present an active approach for handling the user behavior factor.

3.9 P2P Streaming QoS

In theory, P2P based streaming designs and simulations provide a promising alternative to server-based streaming systems both in cost and scalability. In order to examine what made P2P streaming less popular than its alternatives around the world, some evidences about practical problems with these networks should be found.

One of the most promising P2P streaming networks was Joost [24]. Joost suffered from severe QoS problems that were reported by its users such as connection loss, hiccups [6] and degraded throughput [13] that affected video quality. Joost also failed in broadcasting live events [5]. Recently, Joost finally abandoned P2P completely for a server based solution [4]. Another highly popular P2P streaming network is PPLive [29], which is also reported to suffer [15] from occasional glitches, re-buffering and broken streams.

While in server based streaming services it is possible to solve streaming QoS problems with buffering, the instability of peers' upload in P2P streaming networks require a much larger buffer, which puts QoS in question again for the latency - as even though PPLive offers only modest

low-quality narrow-band P2P video streaming [73], its subscribers experience a latency between tens of seconds [100] to two minutes [65].

For a deeper understanding of what kind of protocols or applications are responsible for upstream load in peers, Darwin [20] and VideoLan [32] streaming servers were installed on 10 different home machines with DSL links and broadcasted from each of these machines to our client machine separately. In parallel, each of these machines were running the applications that were installed on them and used regularly. Using a software based sniffer [34], packets were captured in the uplink and the downlink of each of these machines in a time frame of 24 hours, and then analyzed the log. First the streaming throughput received from each machine was calculated while there are no other processes running in parallel. Then the users were allowed to use the machines regularly and the system logged what protocols and processes were running when the average throughput of the streaming was reduced.

Figure 3.7 presents the applications that degraded the streaming throughput the most and their fraction in the total time when the streaming's quality was reduced. Over the time when these processes were allowed to run, lengthy latency, delay and hiccups appeared. Notice that the top applications that affected streaming were P2P file sharing applications and mail clients that sent large attachments. Yet, other users might use other applications, therefore one cannot rely on adjusting the solution for specific protocols, and an appropriate solution must face different user behaviors, different processes and protocols with no prior knowledge about the structure of protocols used by a specific user.

Collaborium already presented a solution for stabilizing throughput, but unlike file sharing applications that are left running in the background even when the user is not actively using its computer, in streaming applications the situation is different. Streaming applications are running mostly when the user is actively using them. When the user is no longer interested in watching media content, the application is turned off, thus the chances of finding free peers that are connected to a streaming network are low.

Thus there is a need for a solution that will address the problem of user behavior in streaming P2P networks and provide a mechanism that is able to perform the following:

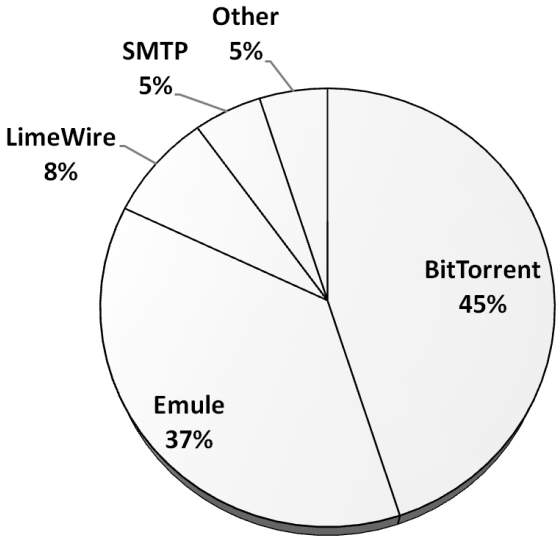

Figure 3.7: Top applications that affect streaming QoS and their relative fraction in time

1. Absorb the instability of source peers and allow stable throughput at the client side
2. Perform well on new protocols with no prior knowledge about protocol structure

3.10 Maxtream

Maxtream is a machine learning based solution that actively predicts load in the uplink of streaming peers and coordinates source peers exchanges between peers that suffer from buffer underrun and peers that enjoy satisfactory buffer size for coping with future problems. Following is a detailed explanation of the key elements of the system.

3.10.1 Behavior Aware Streaming Concept and Mutual Source Exchange

Figure 3.8 illustrates the concept of user behavior aware streaming. N_1,N_2,N_3,N_4,N_5 and N_6 represent nodes/peers in a streaming network. While there are numerous topologies for streaming networks such as trees or mesh based, the design is not limited to a specific topology. As such,

N_1, N_2 and N_3 are unnecessarily at the same level on the graph. Notice that the throughput that N_3 provides to N_5 is high and stable. This high throughput allowed N_5 to fill its buffer over time, therefore N_5 is immune to network problems to some extent. The larger buffer N_5 accumulated, the bigger problem it can absorb without affecting its user experience.

Now let's assume that due to earlier problems, N_4's buffer was reduced and is almost empty, such that its user can still watch a movie with no interruption, but a further problem in the future will empty the buffer and cause a hiccup. As can be seen on the link between N_2 and N_4, N_2 begins with providing a stable throughput for N_4, but later on at time $t_p - \epsilon$ the user at N_2 opened a file sharing P2P software or any other process that consume upload bandwidth. A few seconds later, if nothing else happens, at time t_d the throughput between N_2 and N_4 will be dropped and become unstable due to the new software/process. In a regular streaming network - the buffer on N_4 might get emptied as N_4 does not have enough time to connect to an alternative source.

Maxtream's agent that runs on N_2, predicts at time t_p that it will soon have to share it's upload bandwidth with another process, therefore it immediately notifies N_4 to replace a source. N_4 then disseminates a request for source to his neighbor peers. N_5 receives N_4's request and since its buffer is full, it has a better chance to absorb the drop in throughput of N_2 without completely emptying its buffer; therefore N_4 and N_5 exchange sources, thus N_4 did not experience a hiccup.

In case that the drop in the upload throughput of N_2 will not recover, in time, N_5's buffer will continue emptying as well, and when reached to a level that requires action, N_5 will disseminate a request for an alternative source from another peer that gain a large buffer, or even switch sources with N_4 again if in the meantime N_4 managed to gain a large buffer.

In the experiments, the request for alternative sources was disseminated using a Gnutella like approach - by sending the request to the peer's neighbors, the neighbors forward the request to their neighbors and the mechanism continues in this manner up to a certain predefined radius. Among the peers that receive the request, the peers that are potentially suitable are the peers that already filled their buffer enough to cope with a temporal drop. This approach also preserves the scalability of the networks as the changes in the network's topology are local. Yet, this Gnutella

like approach can be switched with any other message passing mechanism that already exists on each P2P network.

3.10.2 Faulty Source Hopping

The key motivation for mutual source exchange in Maxtream is the potential of a "strong" peer (one that has enough cache or buffer size) to absorb problems in sources that experience a temporal drop in their throughput. As mentioned above, if the drop in throughput lasts long enough to endanger the buffer of the previously strong peer, a second source exchange will be needed. Moreover, while the drop still exists - the "faulty" source travels between "strong" peers. Yet, if the drop does not recover eventually, there is no interest in it as a source as it only empties its client peers' buffers. Thus, a mechanism that will limit the journey of a faulty source is required. An optional solution for this problem is by "pushing" the faulty source to the borders of the network's topology - for example, in a tree based network, this source should eventually become a leaf and cease to serve as a source. Similarly, in a mesh faulty sources should be pushed to the bounds of the network. A counter that counts the exchanges of each source due to drops in a given time frame is being used and in case it crossed a predefined threshold, the faulty source is moved one level down in the tree (or one step towards the bounds in a mesh) and the counter is reset. If the drop continues, eventually the faulty source will become a leaf.

Another direction of exploiting faulty sources is to employ them as sources for a lower grade of service - by having 2 or more different grades of streaming networks - High Definition and Standard Definition. Sources that were pushed down in the HD streaming sub-network up to some threshold of levels, will move to serve as sources for the SD streaming sub-network and sources that show high stability as sources in the SD sub-network will move to the HD sub-network.

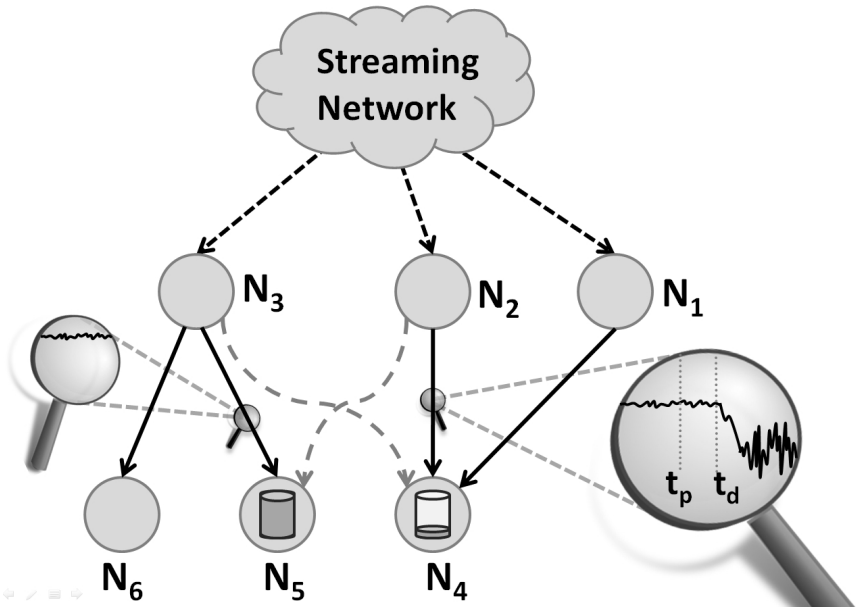

Figure 3.8: Maxtream concept chart

3.11 Testing Maxtream

3.11.1 Streaming Experiment

Two machines were installed - A and B with a proprietary streaming protocol and transmitted from A to a client machine C. All machines were connected to the Internet using a broadband connection of $1.5Mbps$ download and $0.5Mbps$ upload. While A trasmitted to C interruptions were initiated by running various applications (eMule, BitTorrent, Outlook mail client) at specific time periods using a predefined script at A. Notice that for all of eMule tests, the obfuscated version was used. These applications were selected as they consume upload bandwidth. The file that was transmitted was a movie file of $13MB$ that lengths 253 seconds. While the movie was playing in C, the hiccups that were experienced while watching the whole movie were counted. This experiment was repeated with 2 different settings. In Figure 3.9 the Regular case represents a strategy where cache is collected in the streaming buffer up to the defined buffer size and then let it play regularly. In the Maxtream case the agent was installed on A and notified C to switch

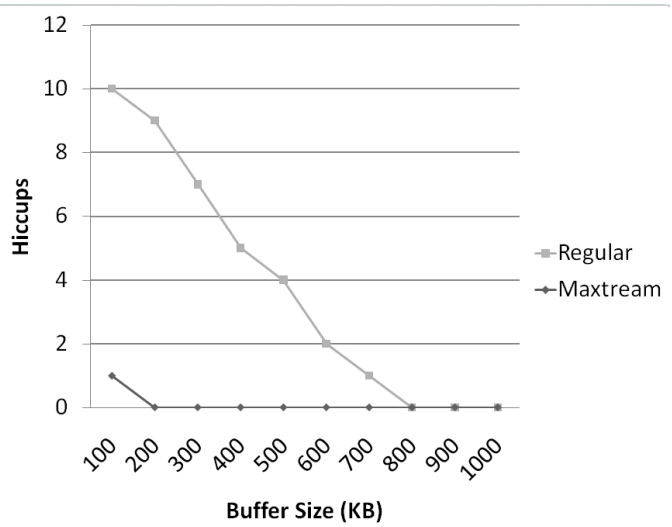

Figure 3.9: Hiccups count per streaming buffer sizes under initiated interruptions

to B when our algorithm predicted an upcoming load in A. The interruption time was limited (by closing the applications after a few seconds) and in the case of Maxtream C was instructed to switch back to A in order to test the next interruption. As can be noticed, for the current settings and the given interruptions, on buffers larger than $100KB$ Maxtream experienced no hiccups, while the Regular case demanded much larger buffers of $800KB$ to reach a case of no hiccups. Figure 3.10 shows the throughput received by the streaming client at C for the first minutes. Notice that on time t the first interruption was activated- by simply launching an eMule client at A. Notice that in the case of Maxtream, C predicted the upcoming load successfully and switched its source to B instead of A while maintaining a stable streaming throughput.

3.11.2 Prediction Accuracy

Figure 3.11 presents the time between the prediction and the beginning of the load in upstream per each of the leading protocols. Notice that there is a time range of between 3 and 6 seconds to alert a client for replacing a source - which enables it to completely evade the upcoming load before it begins.

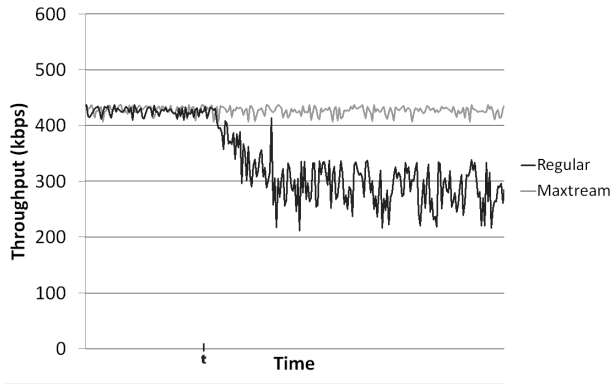

Figure 3.10: Streaming client's throughput

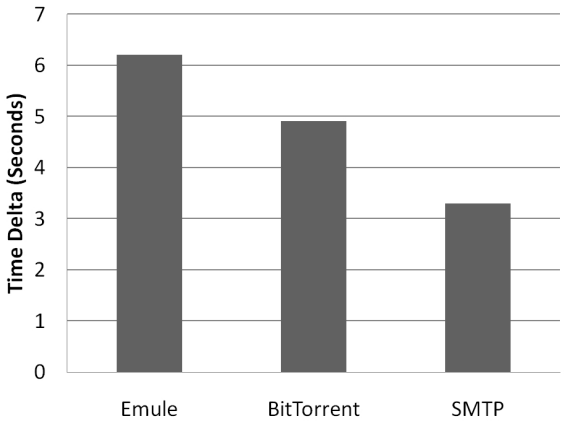

Figure 3.11: Time between prediction and load

Chapter 4

Application Behavioral Patterns

4.1 Predicting Application Behavior

This thesis began with analyzing Traffic Behavioral Patterns, where our knowledge about the flow of data in a P2P network was utilized in order to modify the traffic patterns for solving localization and bandwidth optimization problems.

Then the influence of the user on the traffic was presented, and the solution proposed employed machine learning for extracting User Behavioral Patterns and predicted user behavior for improving the effectiveness of the solutions for the previously presented problems.

While user behavior has its implications on the final traffic by controlling the applications installed in his own machine, the actual packets being sent are subject to the commands of each application. If it's possible to learn the behavioral patterns of each application utilizing its own commands, it will be possible to predict its behavior with higher accuracy.

A running application can be seen as an entity that supports various activities - such as sending email, downloading a file, browsing a web page etc. In each activity, the application calls a sequence of actions - for example if the activity is sending an attachment, then one of the first actions will be an action that reads the attachment file into the memory; later in the sequence of actions there will be an action that sends fragments of that file as network packets. The Application Behavioral Patterns of a specific application is the set of sequences that reflect

its different activities.

Being able to predict resource consumption events can be beneficial in resource management of cloud environments, where applications running in the cloud can be learned for patterns that come before the application demands extensive resources and let the resource management transfer the "problematic" application to a different machine, lowering the risk of resource hogging for other applications running on the same machine. In addition, by learning a subset of resource consumption events such as network use can help us in building behavioral signatures for encrypted protocols, or in the case of monitoring disk routines it can help us building better caching algorithms that can predict which memory blocks to offload and reload due to the correlation of their consumption patterns together with other system events.

In this chapter Seekuence is being presented - a solution that learns the behavioral patterns of specific applications by analyzing the sequence of actions taken by the application. This chapter covers a description and analysis of the algorithm, and then discuss two applications for the same algorithm: encrypted protocol inspection and resource consumption prediction.

4.2 Seekuence Related Work

Some research works try to learn application and system behavior for resource management: Some papers map resource usage based on design elements and specifications [81,41]. This approach is problematic when the system aimed to be learned already exists, or the complexity of the software which may be composed of multiple processes and threads is high for building a model. Other papers focus on specific systems or rely [39,55] on measurements from an existing system and do not present a solution for prediction the behavior of an application but only offer a model or a schematic view [38] for resource consumption. The above papers focus mainly on the design guidelines for creating new software systems, and lack the ability to predict resource consumption live on an existing software. In [80], a monitoring infrastructure that extracts formal, indexable, low-level system signatures is presented, by embedding kernel function calls into vectors; this is done by counting the number of times each kernel function was called during a given time-interval,

yet this statistical approach might miss cases where the order of measured features in a sequence is critical.

In [49], a workload generator that extracts statistics from production traces of MapReduce jobs for better energy efficiency in datacenters is presented. While it can help in better job scheduling using execution time predictions, the approach of the paper is based on pre-measured inter-job arrival time, the input and output data size statistics to characterize MapReduce workloads. Another paper [58] developed a statistical method to predict execution times of MapReduce jobs.The authors of [89] proposed an approach for predicting the CPU utilization of applications running on MapReduce. Yet, all of the above papers do not take an active approach for the running application, that might change its resource allocation patterns. Instead, these papers assume a model where the client that runs the application needs an estimation of the resources required for running the application, taking peak times into account in advance. [9] presents a prediction solution using machine learning but is limited to SQL queries only.

As one of the applications of our solution is identification of applications that use encrypted traffic in the network, relevant academic papers in this field are reviewed here as well. Usually, the process of protocol identification is based on deep packet inspection (DPI) techniques, such as port based classification (for example, port 21 usually represents FTP traffic), payload string matching [92] (such as "GET" in HTTP traffic) and even IP addresses that are known to serve specific applications. Yet, in the recent years many applications are increasingly using unpredictable port numbers [40], and employ encryption and obfuscation on packet content - preventing payload based identification as well. In [82], the authors bring an extensive survey of techniques for Internet Traffic Classification using Machine Learning, covering 17 papers, that use different methods such as Expectation Maximization, Nearest Neighbour, Naive Bayes, Markov chains and K-Means. All papers use similar statistical features such as the total number of packets in a session, packet length statistics (min, max, mean, standard deviation), mean inter arrival time, and port values. None of the papers use sequential sets of values based as features, but only statistical features.

Skype is one of the industry's great challenges for identification. In [96] a relayed traffic

of Skype is analyzed, and a detection method is proposed based on metrics that characterize multimedia relayed traffic. The authors of [46, 36] used statistical measures such as minimum, maximum, median, lower and upper quartiles, mean and standard deviation over Skype's packet length and inter arrival time; yet they require a window of 5 seconds or more to identify skype - which might be worthless for DPI purposes. In [45], the authors present a Chi-Square test for leveraging on randomness introduced by the obfuscation techniques used by Skype.

4.3 Seekuence

4.3.1 Algorithm Phases

First a conceptual algorithm that explains the main ideas behind Seekuence is presented. Following this, an improved version of the algorithm is presented and performance issues are discussed. For the algorithm description, please refer to Figure 4.1. The conceptual algorithm is composed of 3 phases: Init, Scan and Update.

Init Phase

In this phase the data for the algorithm is being prepared. The algorithm accepts a list L of input sequences. A sequence is an ordered list of measurements. For example, if one wish to learn the patterns of packet lengths in TCP session of a specific protocol, then each sequence will list the lengths of all packets per session - thus each sequence appears as a set of ordered numbers. Notice that the sequences are unnecessary of the same length, as in reality TCP session lengths are diverse.

While in the above example the sequence is limited to be a list of objects of the same type (packet length values) - the actual algorithm is not limited to just one type of object. For example: there may be sequences that contain both packet lengths and packet inter-arrival time measurements, for some or all of the sequences. In the case of analyzing the access patterns of an application to the resources of a machine, sequences may be composed of different measurements: disk read operation with length, disk write with length, memory read or write, network access

etc. To keep it simple at this stage, the following phases will be analyzed for sequences of one type of object.

Scan Phase

In this phase the similarity between any pair of sequences in L is calculated. The main idea here is an iterative algorithm that greedily merge the pair of sequences that are the most similar into one cluster. In order to be able to find this pair, in the first scan, an $L \times L$ matrix is created where each cell contains the longest common subsequence of the relevant pair.

The similarity function, $CALC\_LCS$ is based on an adaptation of the *Longest Common Subsequence (LCS)* algorithm [66] for lists of integers instead of strings. A matrix of all pairs of input sequences with their similarity is created. Notice in the algorithm that per each pair of sequences (j, k), the similarity is calculated between j and k only once (thus it actually modifies less than half of the matrix). During this scan, the solution also maintains the largest *LCS* in the matrix, as it's out goal to merge the pair that is most similar after the scan phase.

Update Phase

In this phase, the pair that is most similar (found in the scan phase) is merged into one cluster. This is done by "canceling" one of the pair items from the matrix; in this specific case the larger item is picked (for no particular reason) to be canceled (k) - by marking its related matrix cells with $NULL$. This is done instead of reducing the matrix size by one row and column due to the time complexity of altering the matrix data structure. As for the other item in the pair that was not canceled (j), for the future steps it will represent the cluster $[j, k]$, thus Seekuence recalculates the *LCS* between the new sequence of j $(CALC\_LCS(j, k))$ and any of the other sequences in the matrix.

The algorithm calls in each iteration to the Scan Phase and the Update Phase repeatedly, until a specific termination term is met. The termination term should be adapted to the problem one wish to solve. For example, if one looks for patterns of packet lengths then he might set a minimum threshold on the largest found *LCS* - for instance: at least 3 packet lengths, as patterns

smaller than 3 packet lengths are not meaningful and might result in lots of false positives from other protocols that happen to share the same small sequence of packet lengths. If one looks for patterns of packet inter-arrival times of a protocol, he might set a minimum threshold on the time difference between the first item and the last item of the calculated sequence in the *LCS* - as very short time differences are more sensitive to network infrastructure delays rather than representing inter packet delays that are characteristic to the specific application.

An illustration of the process can be seen in Figure 4.2. In this sample $|L| = 20$ input sequences are given. In the first iteration of the Scan Phase, Seekuence calculates the cells of the matrix; For example: sequence no' 1 and sequence no' 2 have 4 items in common : $9, 17, 3, 25$, thus the similarity between these sequences is 4 items. Yet, the sequences 1 and 20 have 6 items in common, which is the largest subsequence among all pairs in the matrix, thus in the Update Phase Seekuence merges sequences 1 and 20 into one cluster. Sequence 20 will be cancelled and the similarity between the new clustered $[1 + 20]$ sequence and all other sequences is updated accordingly. The next largest similarity is between sequence no' 2 and the clustered $[1 + 20]$ sequence, thus Seekuence will cluster these sequences together.

Tree Pruning

If all merge operations are logged, the whole process can be presented in a dendrogram tree (see Figure 4.3) where each leaf at the bottom of the tree represents one sequence ($|L|$ leafs in total). Upon each merge of sequences a cluster is created which is represented by a father node for the leafs of these sequences. Once this dendrogram is ready, one may decide on the right patterns according to the problem that he wish to solve. For example, one strategy of selecting the list of patterns depicted here is simply pruning the tree at different levels for different number of patterns. Less patterns may cause false positives for identification problems as the small size of each pattern might not be enough to be unique for our class. More patterns may have performance issues if our application must test each potential pattern.

Time Complexity and Improvements

The major problem with the algorithm described above is with its performance. In the first iteration of the Scan Phase Seekuence finds the *LCS* of up to $\frac{L^2}{2}$ cells which sums up to $O(LCS \times L^2)$. For a pair of sequence of sizes m and n, the *LCS* can be done in $O(mn)$. In further iterations (up to $L-1$ iterations if the termination term is not embedded) the pair of sequences with the highest similarity has to be found, by running over all cells which will cost us $O(L^2)$ and then recalculate the *LCS* for one row and one column, such that each iteration costs $O(LCS \times L + L^2)$. Taking $L-1$ iterations, this all sums up to $O(L^3)$ (even more if *LCS* is taken into account) - not very encouraging if $|L|$ can be very large for some applications.

The main issue with performance is the fact that per each iteration Seekuence has to run over all cells in the matrix for finding the most similar pair of sequences. Figure 4.4 presents an improved version of our algorithm - this time using a sorted list. In this version Seekuence manage a sorted list, initially containing all pairs of sequences and they calculated *LCS* size. As in any of the next phases one wish to "cancel" one sequence from the list (which is represented by up to $L-1$ entries in the sorted list), Seekuence also manage an array of size $|L|$ where each entry in the array lists pointers to the entries of the relevant element in the sorted list. Thus, in the Scan Phase Seekuence set the pointers from the array to the entries of the sorted list. This way when Seekuence will cancel a sequence, it will not have to scan the whole list, but only use the pointers from the Array entry of this sequence.

In the update phase of the improved version Seekuence simply pick up the last item in the sorted list (representing the pair of sequences that are most similar among all pairs) and then merge them into one cluster and cancel one of the sequences as performed in the original version.

Building the sorted list costs $O(\frac{L^2}{2} \times LCS + \frac{L^2}{2} \times \log(\frac{L^2}{2}))$. Upon each update phase Seekuence access the elements of the sorted list using the pointers from the array, thus the cost per iteration is only $O(L \times LCS)$, summing up to $O(L^2 \times LCS)$ for all iterations. Thus, the improved algorithm time complexity is $O(L^2 \log L \times LCS)$.

While the new algorithm improves performance in order of magnitude, a large list of sequences still results in performance issues. In the experiments these issues were solved by selecting a

random subset (one tenth) of the list of sequences and calculating its patterns. Then add another random subset from the list (still one tenth) and examine if new patterns are found. This process was repeated until the list of patterns is stabilized and no substantial (over 10% change) modifications for the most popular patterns are found. If there aren't any significant modifications additions after choosing additional subsets, this means that the result is stable and working with the whole dataset is not necessary.

4.3.2 Sequence Similarity

In order to compare a pair of sequences in Seekuence, the longest common subsequence(LCS) algorithm [66] is being used. Instead of characters, numeric values are used (such as packet length), and Seekuence simply compares the difference between these values. If the difference is less than a predefined threshold (that was set to 5 in our solution) Seekuence still interpret the comparison as if the values were equal, and once it clusters sequences together, Seekuence saves the bounds of the range for future comparison to other sequences. This allows us to support value ranges. For example, in our solution the following sequences: $\{17, 1, 2, 3, 10\}$ and $\{82, 1, 3, 14, 20, 51\}$ have the common pattern of $\{1, 3, [10 - 14]\}$ and not $\{1, 3\}$. When the clustered sequence will compare to other sequences in further steps of the algorithm, Seekuence will compare the values of the new pattern to the ranges of our pattern. This allows us to address cases where the actual value of the measured feature is continuous, as in the case of packet length in Skype. This allows us to create less clusters, with more samples in each cluster. The amount of clusters is critical in real time applications such as protocol inspection in wire-speed, where one wish to have as little comparisons to our patterns as possible.

4.4 Solutions

4.4.1 Protocols Identification

One of the challenging applications that sequence can be used for is behavioral protocol identification. Instead of using traditional deep packet inspection (DPI) techniques such as port based classification or payload string matching, Seekuence looks for patterns of packet length sequences per session. This approach makes sense as packet size sequences were already presented in this dissertation for eMule in Collabrium, but this time instead of collecting statistics on the appearance of specific packet lengths Seekuence will find complete repeating sequences of packet lengths - and give significance to the order of packet lengths.

Traffic of eMule, BitTorrent, SMTP and Skype was captured, each covering typical usage scenarios (such as downloading a file, searching for content, sending mail, etc) and over 4Gb per each one. In addition, 20Gb of mixed traffic were captred and let 3 different users run any application including the four applications aimed to be identified, yet, the traffic was captured per each process using a layered service provider (LSP) Windows Driver, that allows us to monitor which packets were sent to or from what process in the operating system; this will later allow us to compare the identification of our solution with the true protocol identity.

Then Seekuence learned packet length sequences per each protocol in separate, using the traffic that was captured per each protocol. In the resulting dendrogram of each protocol, top 10 patterns that has maximal coverage per protocol were selected, thus the process ended up with 40 patterns of packet length sequences in total.

For testing the accuracy of Seekuence, the test run over the mixed traffic captures and per each session it compares the sequence of the lengths of the first 10 packets to the 40 patterns extracted before. In addition, to compare the accuracy with other alternatives, other open source DPI solutions such as L7-Filter [25] and iPoque's OpenDPI [22] were running over the same traffic.

Figure 4.5 compares the accuracy of identifying eMule, BitTorrent, SMTP and Skype using Seekuence, L7-Filter and iPoque. As can be noticed, Seekuence accuracy rate is substantially

higher than other solutions, reaching above 94.5% for all protocols. Worth noting, is the accuracy of Seekuence with Skype of 98.3% using only 10 patterns that only use packet length. The results with Skype, demonstrate the strength of Behavioral Application Patterns in identifying encrypted protocols, as Skype's payload is encrypted in most cases. Other solutions had very low accuracy on this particular protocol.

Figure 4.6 presents the accuracy growth by adding more patterns in the case of Skype. It can be noticed that Skype is dominated by one pattern that covers about 70% of its traffic, and it takes at least 9 additional patterns to cover more than 98% of the traffic. This is dependent on the structure of each protocol, thus in other protocols Seekuence might need much more patterns to reach such coverage. The false positive of identifying Skype using Seekuence was 2.1%.

4.4.2 Predicting Resource Consumption

One application that can benefit from Seekuence is Resource Allocation in IAAS (Infrastructure as a service) and cloud-computing environments. The key idea is to analyze the access patterns of an application to computing resources such as file system, memory and network activity. Using this analysis one can create a behavior model per each application or process. These behavior models can be utilized for better managing the required resources per each running application and predict resource hogging, excessive delay of a service and other QoS problems.

While the presentation of the algorithm of Seekuence used sequences of packet lengths, the algorithm is not limited to this type of measured feature, and the packet lengths can be replaced with any other numeric value. One interesting value to check instead of packet length is the payload of the packet - presented to the algorithm as sequences of byte values from the payload. This allows us to compare Seekuence accuracy with Collabrium on predicting loads in the upstream.

Collabrium counted occurrences of *byte:value* pairs and statistically looked for repeating patterns using SVM. In Seekuence however, the order of the values is taken into consideration thus the expected accuracy should be higher. The algorithm accepts sequences of *byte:value* pairs directly from the packet's payload. As with the experiments with Collabrium, the first 15 bytes

of the payload of each packet that reside in the vicinity of the beginning of long sessions are taken. Seekuence clusters these sequences and builds a dendrogram tree, and then select a set of patterns that represent the traffic of packets that might predict load events. As the tests let Seekeunce find patterns for the payload, one can also let it find patterns of hybrid features, such as sets of $[payloadbytevalues, filesaccessedlist, activeprocesslist]$ by simply altering the similarity function. In this case Seekuence will run the LCS on each metric in separate and assign weight per each feature. In the experiments the weights that were selected were: 0.5 for payload byte values, 0.3 for files accessed list and 0.2 for active process list.

Figure 4.7 examines various protocols that use the upstream and Seekuence's ability to predict an upcoming load per each protocol. As with the previous test with Collabrium, the same 5 hours captures of activity on each of these protocols were used. Then the test mapped all large sessions (more than $1MB$) and counted the cases where Seekuence predicted a large session successfully. The predictions were made for two settings - network only that relied on packet payload only and a combined prediction that was based on network, file system and a process list.

In this test as well as in the previous one with Collabrium, the benefit of file system activity and process list was minor and it was possible to achieve encouraging results with network-only prediction. The results received with Seekuence were 7.6% higher accuracy than Collabrium for the case of using Network, file system and process list events and 8% higher than Collabrium for the case of using only network. This shows that the added information of order with the sequences of Seekuence provide higher accuracy. Seekuence also presented fairly low False Positive of up to 1.5% in all predictions.

Figure 4.8 measures the time between the prediction and the beginning of the load in upstream per each of the leading protocols, as was done in Maxtream in Figure 3.11. Seekuence performs better and allows 19% more time in average for replacing the sources.

The larger time between prediction and resource consumption events, and the high accuracy, show that Seekuence can be beneficial in resource management of cloud environments. Each application running in the cloud can be actively learned for patterns that come before the application demands extensive resources. When Seekuence predicts an upcoming load, the cloud

resource management module may transfer the "problematic" application to a different machine, lowering the risk of resource hogging for other applications running on the same machine.

```
1 INIT_PHASE:
        Denote L as list of input sequences
        FirstIteration = True
2 SCAN_PHASE:
        largestLCS_LEN =0
        For j = 1..|L| do
          For k = 1..|L| do
          {
            If (j<k)
            {
                        If (FirstIteration ==True)
                                LCS_Matrix[j,k] = CALC_LCS (L[j],L[k])

                        If largestLCS_LEN < |LCS_Matrix[j,k]| then
                        {
                          largestLCS_LEN = |LCS_Matrix[j,k]|
                          largestLCS = LCS_Matrix[j,k]
                          largestLCS_j = j
                          largestLCS_k = k
                        }
            }
          }
        FirstIteration = False
        If largestLCS_LEN < MIN_LCS_THRESHOLD Then
                        EXIT()
3 UPDATE_PHASE:
        LCS_Matrix.FillColumn(largestLCS_k, NULL)
        LCS_Matrix.FillRow(largestLCS_k, NULL)
        L[j] = LCS_Matrix[j,k]
        For i = 1..|L| do
        {
          LCS_Matrix[i,j] = CALC_LCS (L[i],L[j])
          LCS_Matrix[j,i] = LCS_Matrix[i,j]
        }
        GOTO SCAN_PHASE
```

Figure 4.1: Seekuence Algorithm

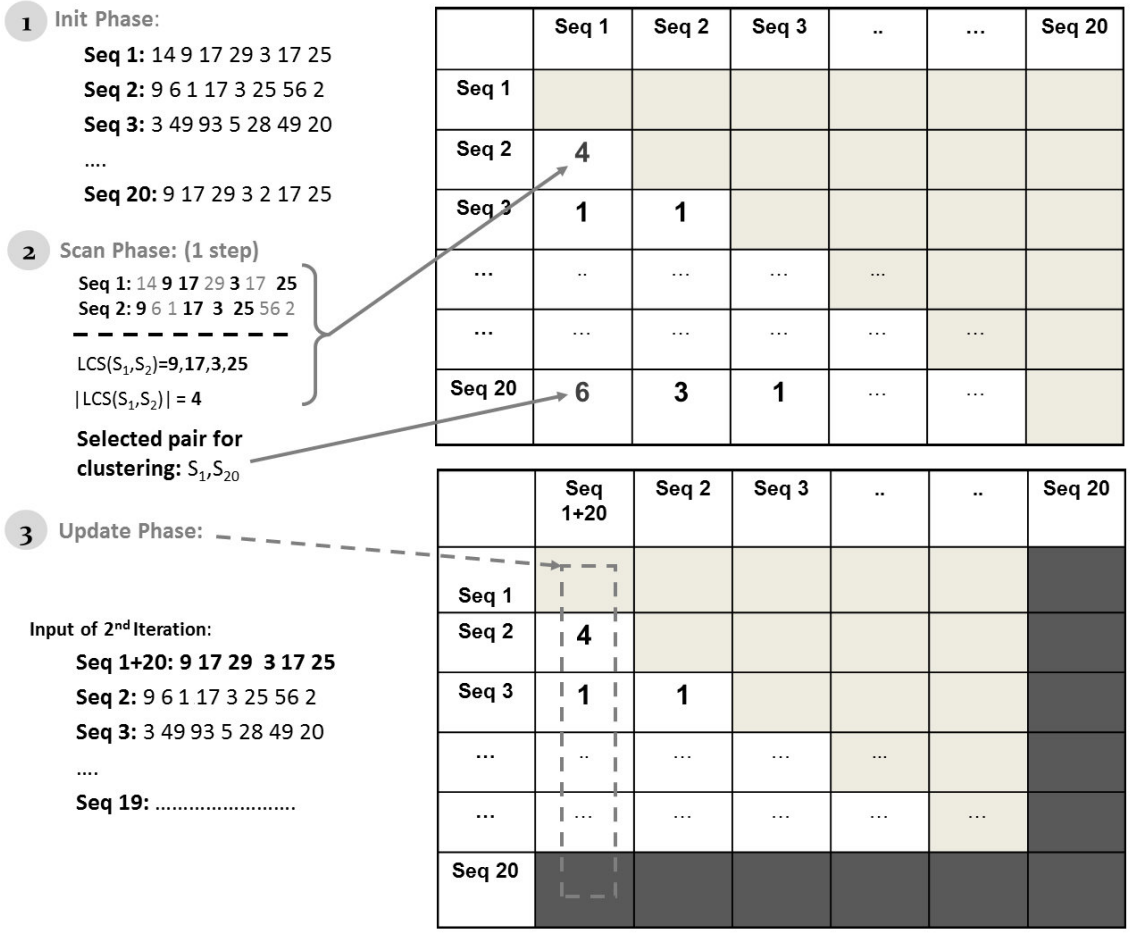

Figure 4.2: Seekuence Algorithm First Steps Illustration

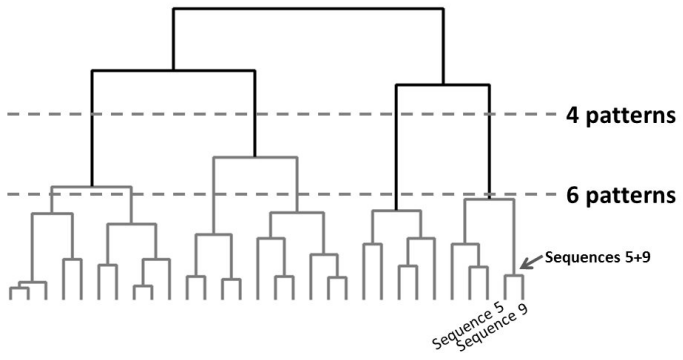

Figure 4.3: Seekuence Tree Pruning

```
INIT_PHASE:
        Denote L as list of input sequences
        Denote Arr as array (size = |L|) of Lists of Pointers
SCAN_PHASE:
        For j = 1..|L| do
          For k = 1..|L| do
          {
            If (j>k)
            {
                    var p = LCS_SORTED_LIST.Add( j, k, |CALC_LCS(L[j],L[k])| )
                    Arr[j].Add(p)
                    Arr[k].Add(p)
            }
          }
UPDATE_PHASE:
        <last_j, last_k, last_lcs_length> = LCS_SORTED_LIST.LastItem
        If last_lcs_length < MIN_LCS_THRESHOLD  Then
                    EXIT()
        For each pointer item p in Arr[last_k] do
        {
             LCS_SORTED_LIST.Delete (p)
        }
        Arr[last_k].ClearList
        L[last_j] = CALC_LCS(L[last_j],L[last_k])
        For each pointer item p in Arr[last_j] do
        {
             <temp_j, temp_k, temp_lcs_length> = LCS_SORTED_LIST.ItemByPointer(p)
             LCS_SORTED_LIST.ItemByPointer(p) = <temp_j, temp_k, |CALC_LCS(L[temp_j],L[temp_k])| >
        }
        For i = last_j+1..|L| do
        {
             <temp_j, temp_k, temp_lcs_length> = LCS_SORTED_LIST.ItemByPointer(Arr[i].ItemByIndex[last_j] )
             LCS_SORTED_LIST.ItemByPointer(Arr[i].ItemByIndex[last_j] ) = <temp_j, temp_k, |CALC_LCS(L[temp_j],L[temp_k])| >
        }
        GOTO UPDATE_PHASE
```

Figure 4.4: Seekuence Improved Algorithm

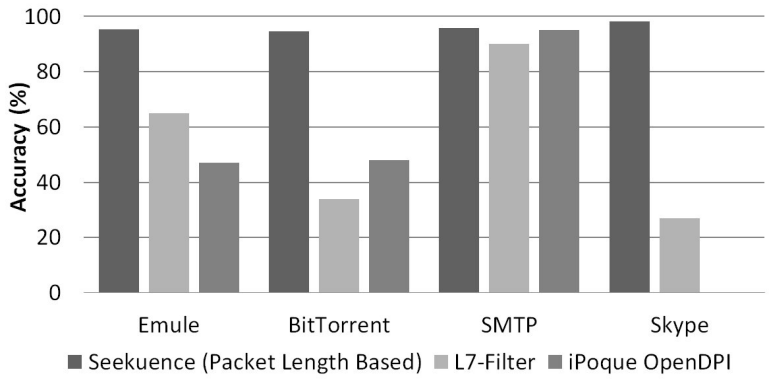

Figure 4.5: Accuracy of protocol identification in Seekuence, L7-Filter and iPhoque's OpenDPI

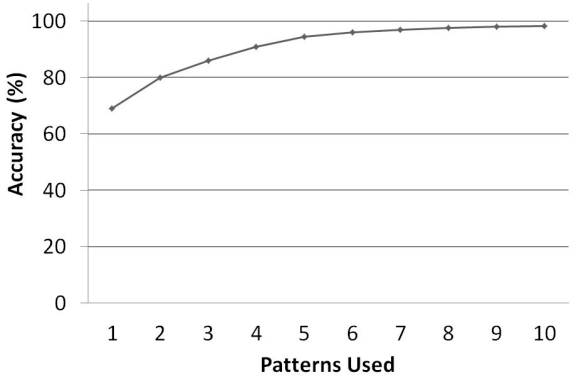

Figure 4.6: Seekuence accuracy of protocol identification in Skype for different number of patterns

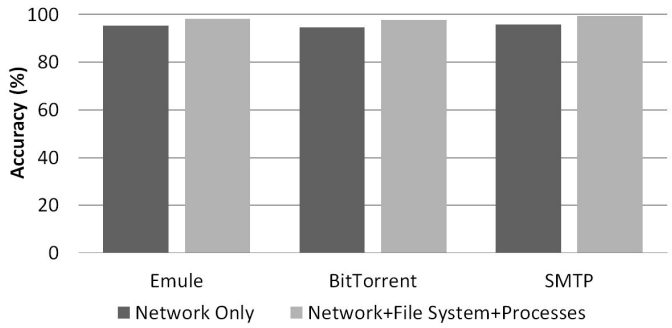

Figure 4.7: Accuracy of prediction in Seekuence for different settings

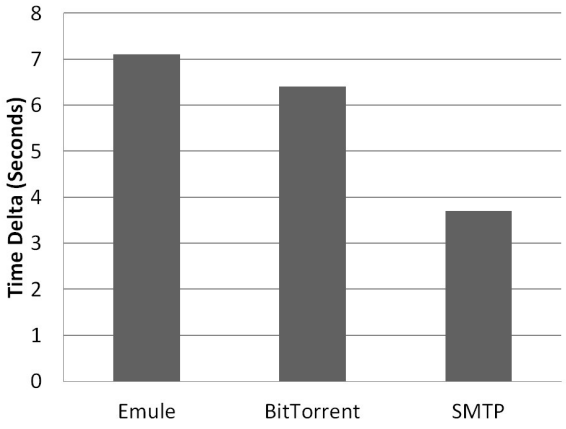

Figure 4.8: Time between prediction and load in Seekuence

Chapter 5

Conclusions and Future Work

The above dissertation addresses behavioral aspects in P2P networks for solving key challenges by utilizing traffic, user and application behavior patterns in P2P networks.

LiteLoad [69] show how modification of traffic behavior patterns can help service providers with localizing popular P2P networks including encrypted ones, without raising the legal disputes of caching solutions or compromising the QoS as in traffic shaping solutions. Collabory [68] modifies the traffic behavior patterns of P2P file sharing networks in order to gain stable download rates and allowing existing P2P networks to efficiently utilize extremely weak or unstable sources. The new notion of Feeders as an evolution of Helper Peers [72] was presented and their contribution for stabilizing the download rate in P2P networks. Substantial improvement is shown for the download rate of P2P file sharing and streaming clients. Collabrium [70] improves the results of Collabory in file sharing networks by taking the user behavior into account, using supervised machine learning algorithms, where the behavior patterns of the user are learned and predict the effects of the user on the throughput of his machine, increasing the download rate of P2P file sharing networks. Maxtream [71] extends the results of Collabrium for P2P streaming networks, and presents a solution for preventing hiccups in P2P streaming networks. Seekuence presents how application behavior patterns can help with predicting excessive resource consumption of applications. Using unsupervised machine learning algorithms Seekuence learn the behavior patterns of each application by utilizing network, processor and file system events. A solution for

identifying encrypted protocols such as Skype behaviorally is presented, without using traditional port or payload based deep packet inspection techniques.

A potential area of improvement is to adapt LiteLoad for mobile service providers. The development of 3rd generation networks such as GSM EDGE, UMTS, CDMA2000 and WiMAX in the recent years has invited a paradigm shift in the consumption of data and internet service within cellular networks as mobile operators began to offer internet access plans that are competitive to fixed line ISP plans. Yet, along with the flow of subscribers towards high speed cellular data links, came also a recognized phenomena in the form of congestion caused by P2P file sharing applications. A recent report [8] presented that mobile data bandwidth usage was increased by 30% globally in the second quarter of 2009 and P2P networks have a lot to do with it. It is noted that the single largest factor leading to cell congestion is P2P which accounts for 42% of bandwidth utilization in the top 5% of cells. The P2P's bandwidth utilization in the average cell is 21%, which is substantial as well. Looking at the cell level, during peak hours while the cell is highly congested it will be beneficial to reduce P2P traffic without degrading the subscribers downlink performance as much as possible, leaving more bandwidth for real-time applications such as VoIP, streaming and Email. On times where the cell is not highly utilized, the interest of the mobile ISP is to use it's internal network of cells to cut the costs of bandwidth on outbound links. LiteLoad may be extended to support the mobile infrastructure where the cells/base stations themselves become bandwidth bottlenecks.

Another direction of improvement is to extend Seekuence to support additional feautures other than packet length, such as packet inter-arrival time(IAT). Packet inter arrival time was already used in Collabrium's monitoring software, yet with Seekuence, IAT will can reinforcing the sequence of features being examined - that may help in creating stronger signatures (composed of multiple features): For example: 1_{st} packet length is 18 bytes , then IAT between 1_{st} and 2_{nd} packet is in the range of 40_{ms}-50_{ms}, then 2_{nd} packet length is in the range of 22-25 bytes. The sequence similarity function may be extended to support the case where values might substitute position as in the case of sessions where packets might change their order in the sequence. In addition, the sequence may be extended to support statistical functions as items in the sequence

such as average of last 30 items packet lengths - meaning that for a flow that contains many packets - every 30 packets will be represented as an item in the sequence that contains the average and the standard deviation of those packets; The same can be done for additional features such as IAT and together can be used for identifying streaming protocols (that have constant IAT and packet lengths for example).

Bibliography

[1] DailyTech story about Bell Simpatico traffic shaping. http://www.dailytech.com/more+isp+ confess+ we+ throttle+ p2p+ traffic/article9544.htm, Nov. 2007.

[2] DailyTech story about Comcast and Peer to Peer. http://www.dailytech.com/comcast+ screws+ with+ filesharing+ traffic/article9337.htm, Oct. 2007.

[3] China mobile p2p traffic localization. http://www.ietf.org/proceedings/74/slides/alto-0.pdf, Oct. 2008.

[4] Joost abandons P2P report in TechCrunch. http://www.techcrunch.com/2008/12/17/joost-just-gives-up-on-p2p, Dec. 2008.

[5] Joost BW problems report by NewTeeVee. http://newteevee.com/2008/03/20/where-to-watch-march-madness, Mar. 2008.

[6] Joost playback problems report by VentureBeat. http://venturebeat.com/2008/11/28/joost-is-loosed-on-the-iphone-if-only-it-worked, Nov. 2008.

[7] Sandvide internet traffic report. http://www.sandvine.com/general/documents/2008 global broadband phenomena - executive summary.pdf, Oct. 2008.

[8] Allot global mobile broadband traffic report. Q2 2009. http://www.allot.com/download/allot_gmbt_report.pdf, July 2009.

[9] *Predicting Multiple Performance Metrics for Queries: Better Decisions Enabled by Machine Learning*, Mar. 2009.

[10] Cachelogic web site. http://www.cachelogic.com, Jan. 2010.

[11] Cisco forecast for internet traffic for 2015. http://torrentfreak.com/cisco-expects-p2p-traffic-to-double-by-2014-100611, June 2010.

[12] Joltid web site. http://www.joltid.com, Jan. 2010.

[13] Joost BW problems report by DailyIPTV. http://www.dailyiptv.com/features/joost-bandwidth-problem-082007, June 2010.

[14] PeerApp web site. http://www.peerapp.com, Jan. 2010.

[15] PPLive glitches report. http://all-streaming-media.com/peer-to-peer-tv/p2p-streaming-internet-tv-pplive.htm, June 2010.

[16] Allot web site. http://www.allot.com, Oct. 2011.

[17] Azureus bad isps wiki web page. http://www.azureuswiki.com/index.php/bad_isps, Oct. 2011.

[18] Bluecoat packeteer packetshaper web site. http://www.packeteer.com/products/packetshaper, Oct. 2011.

[19] Cisco's web site. http://www.cisco.com, Oct. 2011.

[20] Darwin open source streaming. http://dss.macosforge.org, Oct. 2011.

[21] EDonkey implementation - EMule web site. http://www.emule-project.net, Oct. 2011.

[22] IPoque Open DPI classifier. http://www.opendpi.org, Oct. 2011.

[23] IPoque web site. http://www.ipoque.com, Oct. 2011.

[24] Joost web site. http://www.joost.com, Oct. 2011.

[25] L7 filter - application layer packet classifier. http://l7-filter.sourceforge.net, Oct. 2011.

[26] Napa-Wine project technical report. http://napa-wine.eu/cgi-bin/twiki/view/public, Oct. 2011.

[27] Netflix web site. http://www.netflix.com, Oct. 2011.

[28] PeerApp measurements on P2P. http://www.peerapp.com/solutions- managing- transit- link-growth.aspx, Oct. 2011.

[29] PPLive web site. http://www.pptv.com, Oct. 2011.

[30] Sandvine global internet phenomena report. Spring 2011. http://www.sandvine.com/news/global_broadband_trends.asp, May 2011.

[31] Sandvine incorporated web site. http://www.sandvine.com, Oct. 2011.

[32] VideoLan open source streaming. http://www.videolan.org, Oct. 2011.

[33] Vuze bad ISPs list. http://wiki.vuze.com/w/badisps, Oct. 2011.

[34] Windump - TCPdump for windows, http://www.winpcap.org/windump, Oct. 2011.

[35] V. Aggarwal, A. Feldmann, and C. Scheideler. Can ISPS and P2P users cooperate for improved performance? *SIGCOMM Comput. Commun. Rev.*, 37(3):29–40, July 2007.

[36] R. Alshammari and A. N. Zincir-Heywood. Machine learning based encrypted traffic classification: identifying SSH and Skype. In *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, pages 289–296, Piscataway, NJ, USA, July 2009.

[37] N. Andrade, J. Santana, F. Brasileiro, and W. Cirne. On the efficiency and cost of introducing QoS in bittorrent. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 767–772, IEEE Computer Society, May 2007.

[38] T. B. C. Arias, P. America, and P. Avgeriou. Constructing a resource usage view of a large and complex software-intensive system. In *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, WCRE '09, pages 247–255, IEEE Computer Society, Oct. 2009.

[39] T. B. C. Arias, P. Avgeriou, and P. America. Analyzing the actual execution of a large software-intensive system for determining dependencies. *Reverse Engineering, Working Conference on*, 0:49–58, Oct. 2008.

[40] F. B. Baker F and C. Sharp. Cisco architecture for lawful intercept in IP networks. Technical Report RFC 3924, Internet Engineering Task Force, Oct. 2004.

[41] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Trans. Softw. Eng.*, 30:295–310, May 2004.

[42] R. Beverly and M. Afergan. Machine learning for efficient neighbor selection in unstructured P2P networks. In *SYSML'07: Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*, pages 1–6, Berkeley, CA, USA, Apr. 2007.

[43] R. Beverly, K. Sollins, and A. Berger. SVM learning of IP address structure for latency prediction. In *SIGCOMM Workshop on Mining Network Data*, Sept. 2006.

[44] R. Bindal and P. Cao. Can self-organizing P2P file distribution provide qos guarantees? *SIGOPS Oper. Syst. Rev.*, 40(3):22–30, July 2006.

[45] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 37–48, New York, NY, USA, Aug. 2007.

[46] P. A. Branch, A. Heyde, and G. J. Armitage. Rapid identification of Skype traffic flows. In *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '09, pages 91–96, New York, NY, USA, June 2009.

[47] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313, New York, NY, USA, Dec. 2003.

[48] L.-J. Chen, C.-F. Chou, and B.-C. Wang. A machine learning-based approach for estimating available bandwidth. In *TENCON 2007 - 2007 IEEE Region 10 Conference*, Oct. 2007.

[49] Y. Chen, A. S. Ganapathi, A. Fox, R. H. Katz, and D. A. Patterson. Statistical workloads for energy efficient MapReduce. Technical Report UCB/EECS-2010-6, EECS Department, University of California, Berkeley, Jan 2010.

[50] B. Cohen. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems*, 6, June 2003.

[51] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. P-Ring: an efficient and robust P2P range index structure. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 223–234, New York, NY, USA, June 2007.

[52] F. de Asis Lopez-Fuentes and E. Steinbach. Multi-source video multicast in peer-to-peer networks. *Parallel and Distributed Processing Symposium, International*, 0:1–8, Apr. 2008.

[53] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over peers. Technical Report 2002-21, Stanford InfoLab, Mar. 2002.

[54] Z. Despotovic and K. Aberer. A probabilistic approach to predict peers performance in P2P networks. In *8th Intl Workshop on Cooperative Information Agents*, pages 62–76. Springer, 2004.

[55] M. V. Devarakonda and R. K. Iyer. Predictability of process resource usage: A measurement-based study on UNIX. *IEEE Trans. Softw. Eng.*, 15:1579–1586, December 1989.

[56] T. Do, K. A. Hua, and M. Tantaoui. P2VoD: Providing fault tolerant Video-on-Demand streaming in Peer-to-Peer environment. In *Proceedings of the IEEE Int. Conf. on Communications (ICC 2004)*, June 2004.

[57] K. Eger and U. Killat. Fairness in peer-to-peer networks. In A. D. Joseph, R. Steinmetz, and K. Wehrle, editors, *Peer-to-Peer-Systems and -Applications*, number 06131 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, Jan. 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[58] A. Ganapathi, Y. Chen, A. Fox, R. H. Katz, and D. A. Patterson. Statistics-driven workload modeling for the cloud. In *ICDE Workshops*, pages 87–92, Mar. 2010.

[59] P. Garbacki, A. Iosup, D. Epema, and M. van Steen. 2Fast: Collaborative downloads in p2p networks. *p2p*, Sept. 2006.

[60] 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.

[61] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. *Internet Measurement Conference*, Oct. 2005.

[62] Y. Guo, K. Suh, J. Kurose, and D. Towsley. P2Cast: peer-to-peer patching scheme for VoD service. In *Proc. of the 12th Int. Conf. on World Wide Web*, pages 301–309, May 2003.

[63] A. Halme. Peer-to-peer Traffic: Impact on ISPs and Evaluation of Traffic Management Tools. Apr. 2005.

[64] I. U. Haq, S. Ali, H. Khan, and S. A. Khayam. What is the impact of P2P traffic on anomaly detection? In S. Jha, R. Sommer, and C. Kreibich, editors, *RAID*, volume 6307 of *Lecture Notes in Computer Science*, pages 1–17. Springer, Sept. 2010.

[65] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. Insights into PPLive: A measurement study of a large-scale P2P IPTV system. In *Proceedings of IPTV Workshop, International World Wide Web Conference*, May 2006.

[66] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18:341–343, June 1975.

[67] S. Horovitz and D. Dolev. Method and apparatus for managing communications, Patent US 2008/0028055 A, Jan. 2006.

[68] S. Horovitz and D. Dolev. Collabory: A collaborative throughput stabilizer & accelerator for P2P protocols. In *Proceedings of the 2008 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 2008.

[69] S. Horovitz and D. Dolev. Liteload: Content unaware routing for localizing P2P protocols. In *Proceedings of the 5th International Workshop on Hot Topics in Peer-to-Peer Systems (IEEE Hot-P2P)*, Apr. 2008.

[70] S. Horovitz and D. Dolev. Collabrium: Active traffic pattern prediction for boosting P2P collaboration. In *Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, WETICE '09, pages 116–121, IEEE Computer Society, June 2009.

[71] S. Horovitz and D. Dolev. Maxtream: Stabilizing P2P streaming by active prediction of behavior patterns. In *Proceedings of the 2009 Third International Conference on Multimedia and Ubiquitous Engineering*, MUE '09, pages 546–553, IEEE Computer Society, June 2009.

[72] S. Horovitz and D. Dolev. On the role of helper peers in P2P networks. In A. Ros, editor, *Parallel and Distributed Computing, ISBN: 978-953-307-057-5*, pages 203–219. InTech, Jan. 2010.

[73] A. Horvath, M. Telek, D. Rossi, P. Veglia, D. Ciullo, M. A. Garcia, E. Leonardi, and M. Mellia. Dissecting PPLive, SopCast, TVAnts. 2008.

[74] R. Izhak-Ratzin. Collaboration in BitTorrent systems. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 338–351, Berlin, May 2009.

[75] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. In *SIGCOMM '05: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, Aug. 2005.

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

[77] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 75–86, New York, NY, USA, Sept. 2010. ACM.

[78] K. Lakshminarayanan and V. N.Radmanabhan. Network performance of broadband hosts: Measurements and implications. Technical report, University of California at Berkeley, Microsoft Research, March 2003.

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

[80] T. Marian, K. S. Lee, H. Weatherspoon, and A. Sagar. Fmeter: Extracting indexable low-level system signatures by counting kernel function calls. Unpublished manuscript, Jan. 2011.

[81] J. Muskens and M. Chaudron. Prediction of run-time resource consumption in multi-task component-based softwaresystems. In I. Crnkovic, J. A. Stafford, H. W. Schmidt, and K. Wallnau, editors, *Component-Based Software Engineering*, volume 3054 of *Lecture Notes in Computer Science*, pages 162–177. Springer Berlin, May 2004.

[82] T. T. T. Nguyen and G. J. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 10(1-4):56–76, Jan. 2008.

[83] V. N. Padmanabhan, H. J. Wang, and P. A. C. K. Sripanidkulchai. Distributing streaming media content using cooperative networking. pages 177–186, May 2002.

[84] R. S. Peterson and E. G. Sirer. Antfarm: efficient content distribution with managed swarms. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 107–122, Berkeley, CA, USA, Apr. 2009. USENIX Association.

[85] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. One hop reputations for peer to peer file sharing workloads. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 1–14, Apr. 2008.

[86] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips. Tribler: A social-based peer-to-peer system. *IPTPS*, Feb. 2006.

[87] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. Technical Report TR-00-010, International Computer Science Institute, Berkeley, CA, Oct. 2000.

[88] R. Rejaie and A. Ortega. PALS: Peer-to-peer adaptive layered streaming. In *Proceedings of the NOSSDAV 2003, jun 2003*.

[89] N. B. Rizvandi. Preliminary results on modeling CPU utilization of mapreduce programs. Technical Report 665, University of Sydney, School of IT, Sydney/Australia, December 2010.

[90] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329, Nov. 2001.

[91] D. Schlosser, T. Hossfeld, and K. Tutschku. Comparison of Robust Cooperation Strategies for P2P Content Distribution Networks with Multiple Source Download. *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 31–38, Sept. 2006.

[92] S. Sen and O. Spatscheck. Accurate, scalable in-network identification of P2P traffic using application signatures. In *WWW Conference*, pages 512–521. ACM Press, May 2004.

[93] G. Shen, Y. Wang, Y. Xiong, B. Zhao, and Z. Zhang. HPTP: Relieving the tension between ISPs and P2P. *In IPTPS*, Feb. 2007.

[94] L. Souza, F. Cores, X. Yang, and A. Ripoll. DynaPeer: A dynamic peer-to-peer based delivery scheme for VoD systems. In *Euro-Par 2007 Parallel Processing*, volume 4641 of *Lecture Notes in Computer Science*, pages 769–781. Springer Berlin, Aug. 2007.

[95] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, Aug. 2001.

[96] K. Suh, D. R. Figueiredo, J. Kurose, and D. Towsley. Characterizing and detecting skype-relayed traffic. In *Proceedings of IEEE INFOCOM*, Apr. 2006.

[97] D. A. Tran. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE Infocom*, Mar. 2003.

[98] R. van Renesse, M. Haridasan, and I. Jansch-Porto. Enforcing fairness in a live-streaming system. In *Multimedia Computing and Networking (MMCN 2008)*, Jan. 2008.

[99] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[100] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt. Mapping the PPLive network: Studying the impacts of media streaming on P2P overlays. Technical report, August 2006.

[101] J. Wang and K. Ramchandran. Enhancing peer-to-peer live multicast quality using helpers. *Image Processing, ICIP*, Oct. 2008.

[102] J. Wang, C. Yeo, V. Prabhakaran, and K. Ramchandran. On the role of helpers in peer-to-peer file download systems: Design, analysis and simulation. *IPTPS*, Feb. 2007.

[103] J. Wong. Enhancing collaborative content delivery with helpers. Master's thesis, Univ of British Columbia, Sept. 2004.

[104] K. Xu, Z.-L. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. *SIGCOMM Comput. Commun. Rev.*, 35(4):169–180, October 2005.

[105] L. Ying and A. Basu. pcVOD: Internet Peer-to-Peer Video-On-Demand with storage caching on peers. In *DMS*, pages 218–223, Sept. 2005.

[106] Y. Yue, C. Lin, and Z. Tan. Analyzing the performance and fairness of bittorrent-like networks using a general fluid model. In *GLOBECOM*. IEEE, Nov. 2006.

[107] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location androuting. Technical report, Apr. 04 2001.