Collabrium: Active Traffic Pattern Prediction for Boosting P2P Collaboration

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

Abstract

Emerging large scale Internet applications such as IPTV, VOD and File Sharing 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 other processes running on the source peer that consume bandwidth resources.

In this paper we present Collabrium - a collaborative solution that employs a machine learning approach to actively predict 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.

1. Introduction

P2P networks have received significant attention from academia and industry in the past few years for various reasons. 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 rules at the expense of QOS issues like download speed and service continuity, business considerations dictate a different approach where QOS can no longer be compromised.

Unfortunately, popular implementations of P2P based networks in the industry such as Joost and PPLive were reported to often suffer from a variety of QOS problems [22, 3, 7, 4] like broken streams, streaming audio/video hiccups, substantial latency and major delays. While in server based streaming 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 them impractical - as even when networks' policy allowed extremely unbearable latencies of up to 2 minutes [22, 16], users still faced the above problems. In addition, 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 [12, 9]. Alternative non-P2P services such as RapidShare and MegaUpload are rising at the expense of P2P due to its degraded speed [8, 1].

Recent studies have shown that the major factor influencing QOS in P2P networks is the behavior of users at the source peers [17, 15, 20] - 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.

In our previous work [17] we proposed the idea 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, we matched a given client 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, we relied on historical statistics of overlapping online time periods between the client and the feeder. 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 we were able to predict that a potential feeder's uplink is about to be dropped, we could alert the client to select an alternative feeder prior to that drop. This will significantly increase the amount of potential feeders as we will no longer be restricted to bounds dictated by historical statistics of overlapping time periods.

In this paper we present Collabrium - a collaborative solution based on a machine learning approach, that employs Support Vector Machines (SVM) [21] 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. We reinforce our solution 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.

2. Related Work

Machine learning algorithms have been used with P2P for various objectives like evaluating trust and reputation [14], optimizing the path of queries in Gnutella [10], differentiating between P2P-TV protocols by counting packets in a given time frame [6], recognizing anomalies based on IP addresses and port values [23, 19], predicting latency between nodes based on IP addresses [11] and predicting the available bandwidth of an internet path by transmitting probing packets [13]. 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. Other designs made to improve performance in P2P networks are referred in our previous work [17]. We refer to additional related work about P2P stability problems in Section 3.

3. Problem: To P2P Or Not To P2P?

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% of all upstream data in ISPs, it is claimed [8] that 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. RapidShare is already ranked as the 12th web site in global traffic rankings according to Alexa.Com web traffic rating. Another study [1] approves the above and claims that web sites like RapidShare are already responsible for nearly 9% of the Internet traffic in the Middle East and over 4% in Germany. BitTorrent for example - the most popular P2P protocol, suffers from unstable download rates and hardly exploits the available download capacity [12, 9].

One of the most promising P2P streaming networks was Joost, who suffered from severe QOS problems such as connection loss, hiccups [5] and degraded throughput [3]. Joost also failed in broadcasting live events [4] and recently Joost finally abandoned P2P completely for a server based solution [2]. PPLive - Another highly popular P2P streaming network is also reported to suffer from occasional glitches, re-buffering and broken streams [7]. While in server based streaming services it is possible to solve 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 [18], its subscribers experience a latency between tens of seconds [22] to two minutes [16].

The above problems puts 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 [18].

In [17] we 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 [15, 20]. 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.

4. Collabrium

4.1 Behavior Aware Feeders

As we presented in [17], the key concept of Feeders is that we can use 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, we match a given client 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, we can simply use historical measurements or evaluate locality according to the IP address. For ensuring that a feeder will stay online we especially gather statistics of the periods where the feeder was online and maintained a certain threshold of available upload bandwidth; then we match the client 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. We miss feeders that have good connectivity with the client, are currently online but statistically the probability that they will stay online is low or we couldn't realize repeating online periods for these feeders due to lack of history or a varying behavior.
2. Since we look for the maximal online periods, we miss many potential feeders that could serve us for significantly shorter periods.
3. While we can guarantee an effective download rate for our client for long periods, we can't 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 [17], in order to make it much more scalable as well as optimize it to streaming networks, we offer a new approach which sharpens the focus on the real "problem maker" of instability - the user.

In Collabrium, we 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, we employ 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.

Figure 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, we 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 we 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

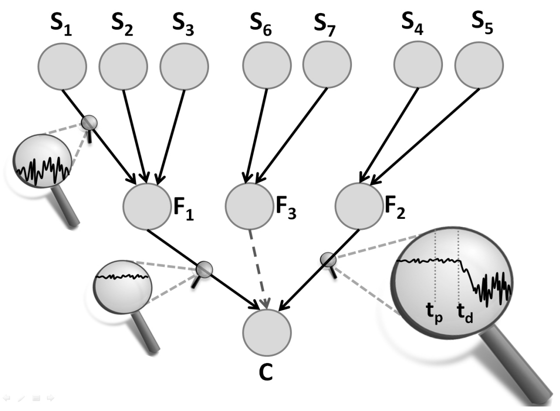

Figure 1. Collabrium concept chart

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 1 can be sources of any P2P network and we don't manage them, but only request for file portions.

4.2 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 we found the network collected data alone to provide sufficient prediction accuracy, we log additional data 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, we only log information in a database for a limited time - while we actually try to learn. 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. We recommend re-running the learning process from time to time, in order to adapt to the user's new habits and trends.

4.3 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

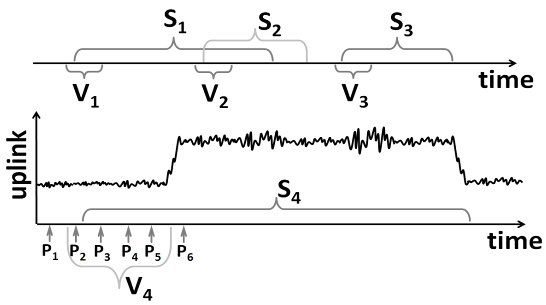

Figure 2. Load Vicinity Pattern Prediction

on a Support Vector Machines (SVM) [21] classification algorithm, yet the assembly of *feature:value* pairs is not straightforward as we elaborate here.

We wish our learning algorithm to link the collected data to the occurrences of traffic load in the uplink. As illustrated in Figure 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 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, we see it as 2 sessions. Notice that sessions might overlap as in sessions S_1 and S_2 but still we can 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 we show the change in uplink utilization due to that session. 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 we can use the packet P_3 and its neighbors 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 we can predict 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 we present different properties that proved to be significant for prediction and their extraction techniques:

Load Vicinity Pattern Prediction: In this method we look at the first bytes (15 bytes were found to be effective) of the payload of each packet that is in the vicinity and extract *feature:value* pairs for SVM so it can learn specific patterns. For example, in eMule's client-client protocol, the 1st byte is always *0xE3* and in the handshake message the 6th is always *0x01*; we mark them as *byte:value* pairs that form a

```
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. Extracting popular *byte:value* pairs

pattern: *1:0xE3*, *6:0x01*. We'd like SVM 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, we can't present SVM these values directly as it will not relate them as discrete values. Therefore, we collect the most popular *byte:value* instances of packets in the vicinities of all sessions while giving priority to *byte:value* pairs that appear in different sessions, as shown in Figure 3.

Finally, we supply the training set for SVM; Each item in the training set contains the following features: Source IP, Source port, Destination IP and Destination port. Then we create a feature per each of the top popular items in *ByteValueList*; i.e. if the most popular *byte:value* pair is *5:0xE3* and the value of the 5th byte of the packet we examine is *0xE3* then we 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 3rd byte of the packet we examine is *0xC2* then we 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 (we found that the top 100 popular yield satisfactory results). We label as *+1* training items that represent packets in the vicinity that contain at least one instance of the top popular *byte:value* pairs. We supply the training set also packets that are not in the vicinity and label them as *-1*. When we run the prediction module to look for upcoming loads in the uplink, we simply propose recent captured packets' properties to SVM with the appropriate features and SVM classifies the packet as leading to uplink load or not.

Packet Size Sequence Prediction: While looking at the data we captured in the beginning of sessions, we noticed an interesting phenomenon in P2P protocols - 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 we

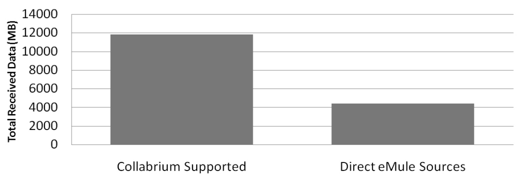

Figure 4. Received data during test periods

noticed some slight differences in the sequence, we can't 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, we relate these values as a histogram, and simply define a predefined number of features (we found 30 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*, we supply the training set 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.

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 we monitor 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 we already used: in the vicinity of the beginning of sessions that raised the load on the uplink, we 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, we notify the client to select an alternative feeder. The smae is done for file system events and precesses

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.

5. Results

We implemented our solution 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. We installed the prediction module on machines B,C,D,E and F that were used as feeders for A. Prior to the week of tests, we allowed our learning module to run for 24 hours on each machine. Along the week of tests, we let A download 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 we deleted the received files to prevent caching for the next test. On the tests that used feeders we initially selected 3 feeders randomly and changed a feeder 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. For all tests we used SVM with Radial Basis Function (RBF) Kernel that was found to perform best for our 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 in nonlinear. After experiencing with different γ and C values for various scenarios, we found that γ of 0.5 and C of 2 provided satisfying results. Figure 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.

In Figure 5 we examined various protocols that use the upstream and Collabrium's ability to predict an upcoming load per each protocol. We captured 5 hours of activity on each of these protocols separately. Then we mapped all large sessions (more than $1MB$) and counted the cases where Collabrium predicted a large session successfully. Notice that in the fourth case, we ran all protocols 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 we were able to achieve encouraging results with network-only prediction. Along all tests we measured the time between the prediction and the beginning of load in upstream, and found that that client has between 3 and 6 seconds to replace a feeder in order to prevent reduction in throughput.

In Figure 6 we experimented different vicinity sizes and measured the appropriate prediction success rate. The leading part of the vicinity (3^rd of its size) is placed before the beginning of a session - to allow prediction using packets

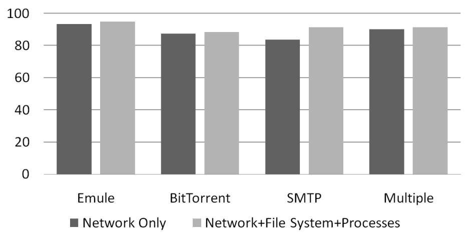

Figure 5. Prediction rate for different settings

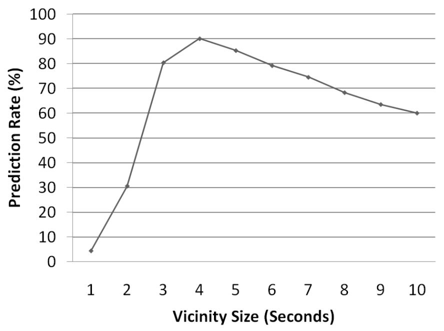

Figure 6. Prediction rate per vicinity size

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 our experiments we experienced minor false positive predictions of up to 2% of the total predictions.

Summary: We presented Collabrium - a collaborative solution that employs machine learning to predict load and stabilize download throughput in P2P networks using feeders. 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 about protocol format.

References

[1] Ipoque internet traffic report. http://www.ipoque.com/userfiles/file/internet_study_2007_abstract_en.pdf.

[2] Joost abandons p2p report in techcrunch. http://www.techcrunch.com/2008/12/17/joost-just-gives-up-on-p2p/.

[3] Joost bw problems report by dailyiptv. http://www.dailyiptv.com/features/joost-bandwidth-problem-082007/.

[4] Joost bw problems report by newteevee. http://newteevee.com/2008/03/20/where-to-watch-march-madness/.

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

[6] Napa-wine project technical report. http://www.napa-wine.eu/twiki/pub/public/documents/napa-abacus-wp3.pdf,.

[7] Pplite glitches report. http://all-streaming-media.com/peer-to-peer-tv/p2p-streaming-internet-tv-pplive.htm.

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

[9] 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, Washington, DC, USA, 2007. IEEE Computer Society.

[10] 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, 2007. USENIX Association.

[11] 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.

[12] R. Bindal and P. Cao. Can self-organizing p2p file distribution provide qos guarantees? *SIGOPS Oper. Syst. Rev.*, 40(3):22–30, 2006.

[13] 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*, 2007.

[14] Z. Despotovic and K. Aberer. A probabilistic approach to predict peers performance in p2p networks. In *In: 8th Intl Workshop on Cooperative Information Agents*, pages 62–76. Springer, 2004.

[15] T. Do, K. A. Hua, and M. Tantaoui. P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment. In *Proc. of the IEEE Int. Conf. on Communications (ICC 2004)*, jun 2004.

[16] 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 *In Proc. of IPTV Workshop, International World Wide Web Conference*, 2006.

[17] S. Horovitz and D. Dolev. Collabory: A collaborative throughput stabilizer & accelerator for p2p protocols. In *IEEE WETICE 4th International Workshop on Collaborative Peer-to-Peer Information Systems (COPS)*, 2008.

[18] A. Horvath, M. Telek, D. Rossi, P. Veglia, D. Ciullo, M. A. Garcia, E. Leonardi, and M. Mellia. Dissecting pplive, sopcast, tvants. 2008.

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

[20] R. Rejaie and A. Ortega. Pals: Peer-to-peer adaptive layered streaming.

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

[22] 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.

[23] 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.