11

On the Role of Helper Peers in P2P Networks

Shay Horovitz and Danny Dolev
Hebrew University of Jerusalem
Israel

1. Introduction

Recent studies in peer-to-peer (P2P) networks present surprising new designs that rely on helper peers. Helper peers, sometimes named as Feeders or Support peers are nodes that do not function as direct consumers or providers of content but are used to collaborate with other peers in the network for a growing variety of benefits.

In File Sharing networks for instance, due to frequent joins, leaves and the characteristic fluctuating throughput of source peers, clients usually download at an unstable rate. In addition, existing P2P protocols tend to ignore source peers that have relatively low bandwidth to offer and practically miss a potentially huge resource. By employing helper peers that are optimal for availability and throughput stability with the downloading client, it is possible to provide a maximal stable throughput even with extremely weak and unstable sources.

Other interesting examples of helper peers in file sharing demonstrated how to integrate helper peers in order to increase the number of sources under flash crowds situations, how to solve the last chunk problem and how to bypass fairness rules for better download rates.

In P2P streaming networks such as live IPTV and VOD, helper peers can contribute in preventing glitches and expanding the dissemination of packets, as well as synchronizing and ordering frames for the clients.

In this chapter we present novel architectures that embed helper peers in order to solve key problems in P2P networks. We discuss the implications and key techniques in each proposal and point the weaknesses and limitations of mentioned architectures.

We present different selection criteria for choosing the optimal helper peers based on theoretic simulations, practical measurements and experiments with popular protocols such as eMule and BitTorrent.

We propose an advanced Machine Learning based design that actively learns the behavioural patterns of peers and leverages the performance of clients by collaborating with the ”right” helper peers at the right time.

Though helper peers gained popularity in P2P research, different works in this field term the same ideas differently and in some cases do not mention each other; this chapter presents the current state of the art in helper-supported P2P networks. Finally, we present future research directions in this field.

2. Background

P2P technology earned its fame throughout the last decade as a result of 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 introduced some promising designs in order to overcome the major challenges that relates to P2P networks – mainly dealing with the Lookup problem but also with security, scalability and performance. The potential of P2P for 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.

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. While P2P is still responsible for more than 60% of all upstream data in ISPs, it is claimed 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 (see e.g. Sandvide 2008 Global Broadband Phenomena (2008)). RapidShare is already ranked as the 17th web site in global traffic rankings according to Alexa.Com web traffic rating. Another study (see e.g. IPoque Internet Study (2007)) supports 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 (see, e.g. Cohen (2003)) for example – the most popular P2P protocol, suffers from unstable download rates and hardly exploits the available download capacity (see e.g. Bindal and Cao (2006), Andrade et al. (2007)).

One of the most promising P2P streaming networks was Joost, which suffered from severe QOS problems such as connection loss, hiccups (see, e.g. VentureBeat report (2008)) and degraded throughput (see, e.g. DailyIPTV report (2007)). Joost also failed in broadcasting live events (see, e.g. NewTeeVee report (2008)) and recently Joost finally abandoned P2P completely for a server based solution (see, e.g. TechCrunch report (2008)). PPLive - Another highly popular P2P streaming network is also reported to suffer from occasional glitches, re-buffering and broken streams (see, e.g. All-Streaming-Media report (2008)). While in server based streaming services it is possible to solve QOS problems with buffering, the instability of peers' upload in P2P streaming networks requires 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 (see, e.g. Horvath et al. (2008)), its subscribers experience a latency between tens of seconds (see, e.g. Vu et al. (2006)) to two minutes (see, e.g. Hei et al. (2006)).

The above problems put P2P technologies in question for commercial system designers. 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 behaviour. 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 (see, e.g. Horvath et al. (2008)).

In Collabory (see, e.g. Horovitz and Dolev (2008)) 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

behaviour 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 becomes significantly unstable. Studies confirm that the major factor that has direct impact on QOS in P2P networks is the behaviour of users at the source peers (see, e.g. Do et al. (2004), Rejaie et al. (2003)). This behaviour leads to fluctuating rate of packets for the client peer that might be reflected by a reduced download rate in file sharing networks or high latencies, delays, hiccups and freezes in streaming P2P networks.

As the existing model of P2P networks failed to provide a stable download speed both in file sharing and streaming, some research papers proposed the idea of employing Helper peers, sometimes named as Feeders or Support peers – peers that do not function as direct consumers or providers of content but are used to collaborate with other peers in the network. In the following sections we will survey different implementations of Helper peers for different applications in P2P networks. We focus on designs that aim to solve the stability problem– as we believe that Helpers will play a crucial role in creating future P2P networks that are competitive with old school's centralized file hosting and streaming systems.

3. Helpers for Service Availability

3.1 Increasing Sources in File Sharing & Multicast

The concept of employing helper peers in a P2P networks was first proposed by Wong (see, e.g. Wong (2004)). 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 Fig. 1). The helper aims to upload each file piece (portion) 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. Its 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 (see, e.g. Wang et al. (2007)). 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 (see, e.g. Pouwelse et al. (2006)). 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. (see, e.g. Wang et. al. (2007)) 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 protocol. Yet, it is claimed that the increased upload contribution only marginally improves download rates in BitTorrent (see, e.g. Piatek (2008)). 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 (see, e.g. Wang et al. (2008)) 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 De Asis Lopez-Fuentes and Steinbach's (see, e.g. De Asis Lopez-Fuentes and Steinbach (2008)) and DynaPeer (see, e.g. Souza et. al (2007)), where helpers take part in a collaboration process for a specific video stream that is managed by a virtual server.

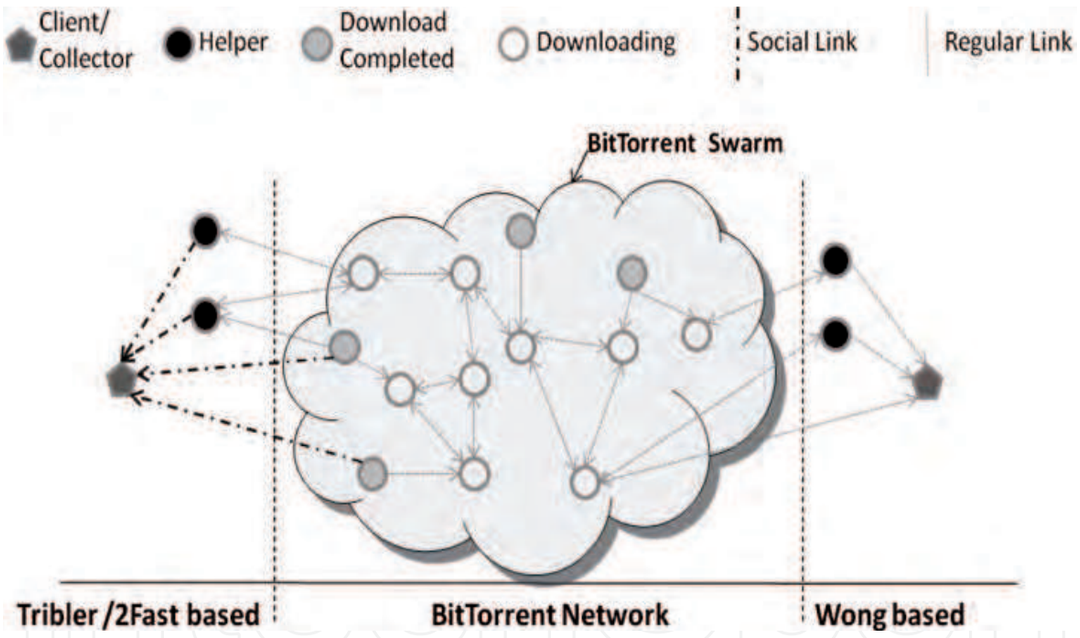

Fig. 1. Wong and Tribler's additions to a BitTorrent swarm-based network

3.2 Social Helpers

In Tribler (see, e.g. Pouwelse et al. (2006)) it was proposed to associate the helpers' contribution with social phenomena such as friendship and trust. In their 2Fast file sharing protocol (see, e.g. Garbacki et al. (2006)), 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 Fig. 1). 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 (see, e.g. Wang et al. (2007)). 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 others might have but the "friends" are not online or running the Tribler client when required. As Wong's work, Tribler did not address the problem of bandwidth stability of a helper peer either. Again, this work's contribution is also limited to BitTorrent-like swarming architectures.

In between Wong's work and Tribler, Guo et al. (see, e.g. Guo et al. (2005)) 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 not necessarily provide any performance gain (see, e.g. Wang (2008)).

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.

3.3 Fairness and Free-Riding

In addition to the anti free-riding solution that was proposed in 2Fast and Tribler, it was shown (see, e.g. Izhak-Ratzin (2009)) 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.

3.4 Key Lookup

In P-Grid (see, e.g. Crainiceanu (2004)) – 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.

4. Helpers for Service Performance

While the availability of content in a P2P network can be increased by employing the techniques mentioned in the previous section, the performance of a peer's service is still directly influenced by the user that operates this peer.
While working on Collabory (see, e.g. Horovitz and Dolev (2008)), we found that the greatest impact on download rate stability is the behaviour 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.

4.1 Feeders

For addressing this problem, we proposed Collabory (see, e.g. Horovitz and Dolev (2008)), where we defined a new type of helper peers that serve as a proxy cache for the benefit of a client peers that wish to download a file; we named these helpers as Feeders. The Feeder stores the file's pieces from several unstable sources and offers 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. Unlike previous works, Collabory intentionally selects the helpers to be optimal for availability and throughput stability with the client by constantly measuring stability factors. 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. Unlike normal helper peers that only assist content delivery, Feeders are employed exclusively as a means of delivering data to the client.
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, we 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, we log feeders' online periods (sessions) and the relevant network use and CPU utilization measurements within these sessions. We 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:

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

where $f_{S_{cpu}}(t)$ and $f_{S_{bw}}(t)$ are the measurements of cpu utilization and consumed upload bandwidth 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 (when the consumer requested to start a new download) is maximized over all other feeders:

$$p = \underset{f}{\text{argmax}} \int_{t}^{t+k} \sum_{i=1}^{n_f} \frac{FA_{f,s}(t)}{n_f} \quad (2)$$

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

In Fig. 2, $C_{regular}$ represents the case of normal file transfer - downloading from *m* sources, each supplyin$\frac{MaxD}{m}$ *bps* g 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 downloads from two sources: the 1st source supplies $\frac{MaxD}{m}$ *bps* and the second source up to ε *bps*. We use the sources that supply ε as for short-term caching to ensure that the feeder peer can always supply $\frac{MaxD}{m}$ *bps* for its client.

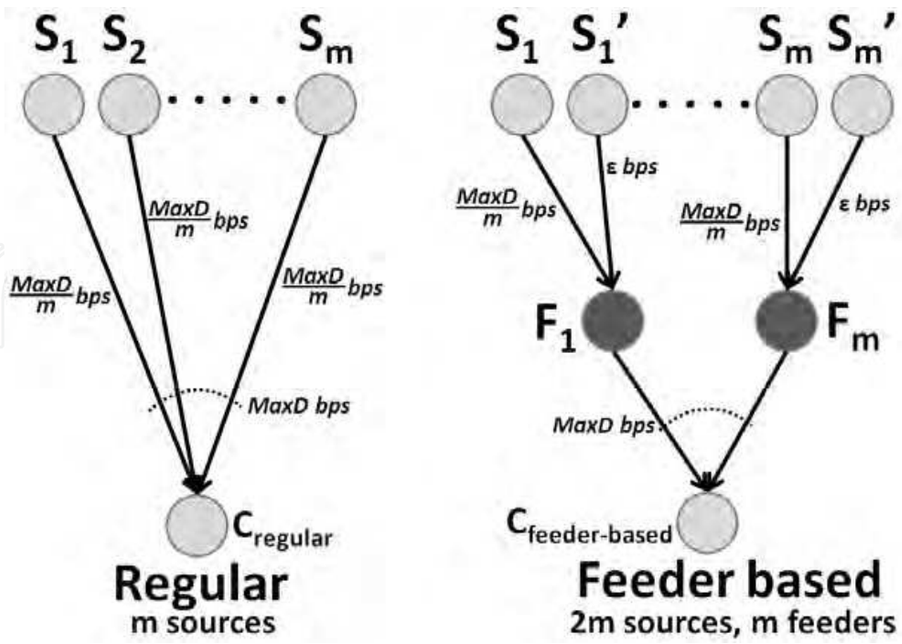

Fig. 2. Schematic view of regular P2P and Feeder based P2P file transfer

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 (Internet link capacity):

$$EDR(C_{regular}) = m(EUB\ of\ each\ \boldsymbol{source}) = mp_s(capacity\ of\ each\ source) = mp_s\frac{MaxD}{m} = p_sMaxD \quad (3)$$

where *EUB* is the effective upload bandwidth.

$$\begin{aligned} &EDR(C_{feeder-based}) = m(EUB\ of\ each\ \boldsymbol{feeder}) \\ &= m(p_f(UB\ of\ feeder\ depended\ on\ its'sources)) \\ &= m(p_f \min(\frac{MaxD}{m}, \sum EUB(feeder's\ sources))) \\ &= \min(p_fMaxD, p_fp_sMaxD + mp_fp_s\varepsilon) \end{aligned} \quad (4)$$

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

$$\frac{EDR(C_{feeder-based})}{EDR(C_{regular})} = \frac{\min(p_fMaxD, p_fp_sMaxD + mp_fp_s\varepsilon)}{p_sMaxD} = \min(\frac{p_f}{p_s}, p_f + \frac{mp_f\varepsilon}{MaxD}) \quad (5)$$

meaning that $C_{feeder-based}$ will download at higher speed than $C_{regular}$ if $p_f + \frac{mp_f\varepsilon}{MaxD} > 1$, which means that $\varepsilon > \frac{(1-p_f)\mathrm{MaxD}}{p_fm}$. Notice that as m grows, a smaller ε will satisfy the benefit of the feeder-based solution. Likewise, if we allow a bigger ε we can use less feeders 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 - meaning that we use a bigger ε 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 we can adjust ε dynamically during the download phase, we can afford 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 we estimate 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.

In Fig. 3, we set the maximum download throughput of all peers 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.
We examine different values of ε to see how it affects the performance of feeders. We set all source peers to behave in a repeating pattern of sending at 80% of their maximal upload bandwidth for 10 seconds followed by additional 10 seconds of sending at full speed. Sources that transmit ε repeatedly transmit 0.8 ε Kb/Sec for 10 seconds and then ε Kb/Sec for the following 10 seconds accordingly. 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 the client accordingly.
Notice that when we set ε to 2.2 the cache content was increasing consistently thus allows the feeder to transmit the client as if it was a stable source.
In this scenario the client received stable download rate of 18.9Kb/Sec.

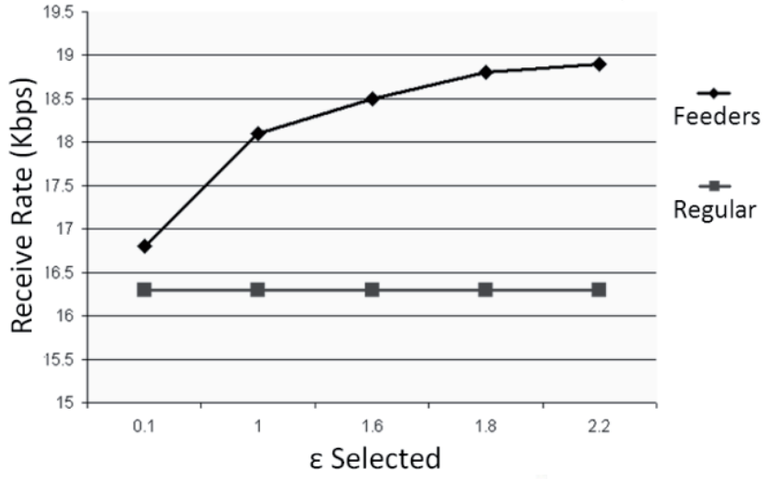

Fig. 3. Feeder-based P2P versus regular P2P with different ε values

We also tested the case of using weak source peers for the feeder (See Fig. 4). For the regular method we set 2 sources of 10Kb/Sec with the behaviour of 80% mentioned above. For the feeder method we set the following different test settings- *A*: 4 sources of 6.0Kb/Sec under 80% behaviour as mentioned above. *B*: 8 sources of 3.0Kb/Sec under 80% behaviour. *C*: 8 sources of 4.0Kb/Sec under 50% behaviour. In all of our tests we gained stable increased rate in the feeder case compared to unstable rate in the regular case.

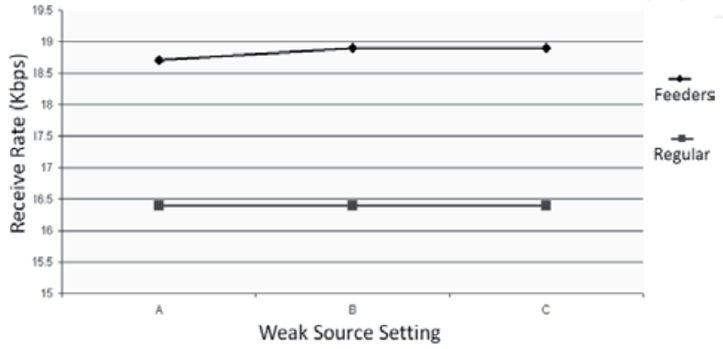

Fig. 4. Feeder-based P2P versus regular P2P with different settings of weak sources

5. Helpers and Machine Learning

In order to guarantee the long service of a suitable feeder, Collabory 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.
Collabory's problems were discussed and addressed in Collabrium (see, e.g. Horovitz and Dolev (2009a)) and Maxtream (see, e.g. Horovitz and Dolev (2009b)). Collabrium is a collaborative solution based on a machine learning approach, that employs SVM - Support Vector Machines (See, Vapnik (1995)) 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 improve the prediction even more.

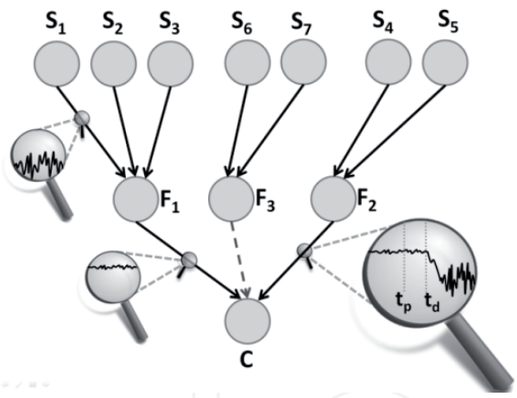

Fig. 5. Learning Feeders in Collabrium

Fig. 5 illustrates the concept of user behaviour 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 represent 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 *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 $t_p - \varepsilon$ 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 its 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 Fig. 5 can be sources of any P2P network and we don't manage them, but only request for file portions.

Following, we discuss the structure of Collabrium that is composed of 3 modules: Monitoring, Learning and Prediction.

5.1 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 behaviour 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.

5.2 Learning Module Design

The learning process 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 classification algorithm, yet the assembly of *feature:value* pairs is not straightforward as we elaborate here.

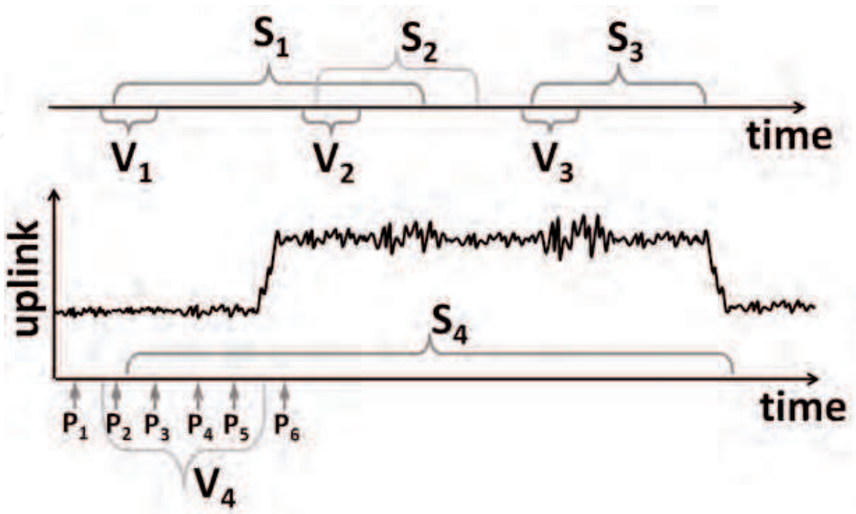

Fig. 6. Load Vicinity Pattern Prediction Concept

We wish our learning algorithm to link the collected data to the occurrences of traffic load in the uplink. As illustrated in Figure 6, 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.

5.2.1 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 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 7.

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.

```
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
```

Fig. 7. Algorithm for extracting popular *byte_value* pairs

5.2.2 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 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 5th packet while in other cases it will be the byte count of the 6th 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.

5.3 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.

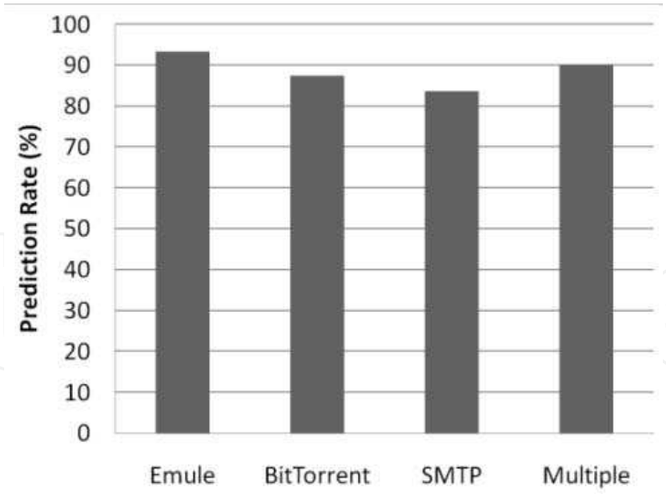

Fig. 8. Prediction success rate of popular protocols

In Figure 8, we examined various protocols that use the upstream and the 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 we 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.
In Figure 9, we measured the time between the prediction and the beginning of the load in upstream per each of the leading protocols.

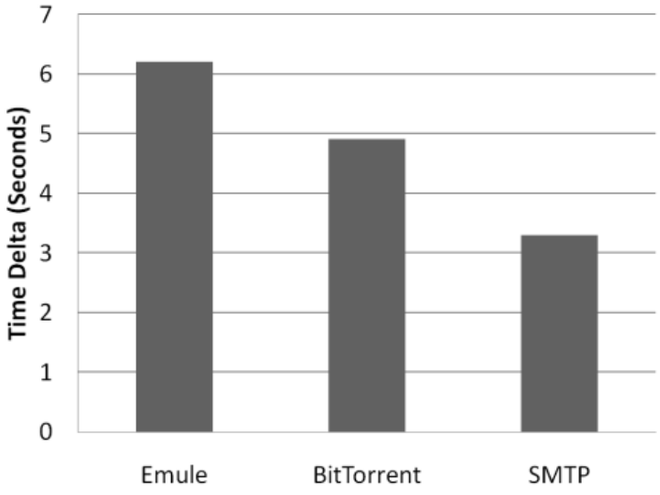

Fig. 9. Time difference between prediction and load (seconds)

Notice that we have 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.

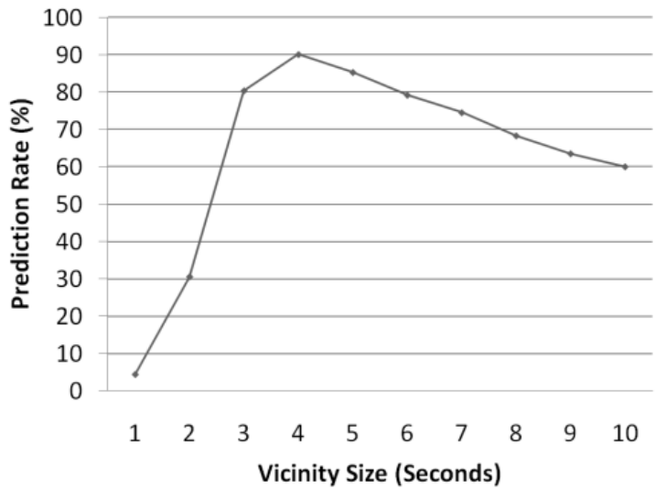

Fig. 10. Prediction success rate per vicinity size

In Figure 10, we experimented different vicinity sizes and measured the appropriate prediction success rate. The leading part of the vicinity (3rd 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 cover enough information to predict an upcoming load with high success rate. In addition, vicinities larger than 4 seconds begin to create more noise than useful information for prediction and accordingly the prediction success rate degrades.

6. Summary and Future Work

In this chapter, we presented the evolution of helper peers in P2P file sharing and streaming networks.

We presented advanced designs of helpers that integrate machine learning for reaching stability in throughput.

We believe that helpers will play a crucial role in the design of future P2P networks, as it enables P2P to compete with both service availability and stability of traditional client-server systems but with much larger scalability. The next required step is to embed and adapt the mentioned ideas onto large scale P2P networks and measure their benefits under different scenarios.

In addition, it will be interesting to analyze different topologies of networks of helper peers. For example, a two-tier helper network might manage 2 different classes of helpers, a hash ring of helpers, a tree of helpers and other topologies.

7. References

All-Streaming-Media report (2008) – PPLive glitches. *http://all-streaming-media.com/peerto-peer-tv/p2p-streaming-internet-tv-pplive.htm*

Andrade, N.; Santana, J.; Brasileiro, F. & Cirne, W. 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

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

Cohen, B. (2003). Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems, 6, 2003*

Crainiceanu, A.; Linga, P. ; Machanavajjhala, A. ; Gehrke, J. ; Shanmugasundaram, J. (2004), P-Ring: An Index Structure for Peer-to-Peer Systems, *Technical Report, Cornell University, 2004*

DailyIPTV report - Joost bandwidth problems (2007), *http://www.dailyiptv.com/features/joostb andwid thproblem-082007/*

De Asis Lopez-Fuentes, F.; Steinbach, E. (2008), Multi-source video multicast in peer-to-peer networks, *IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008*

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

Garbacki, P.; Iosup, A.; Epema, D. & van Steen, M. (2006). 2fast: Collaborative downloads in p2p networks. *p2p, 2006*

Guo, L.; Chen, S.; Xiao, Z.; Tan, E.; Ding, X. & Zhang, X. (2005). Measurements, Analysis, and Modeling of BitTorrent-like Systems. *Internet Measurement Conference, 2005*

Hei, X.; Liang, C.; Liang, J.; Liu, Y. & Ross, K.W. (2006). Insights into pplive: A measurement study of a large-scale p2p iptv system, *Proceedings of IPTV Workshop, International World Wide Web Conference*

Horovitz, S. & Dolev, D. (2008). Collabory: A Collaborative Throughput Stabilizer & Accelerator for P2P Protocols, *Proceedings of the 2008 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp.115-120, 2008, IEEE Computer Society

Horovitz, S. & Dolev, D. (2009a). Collabrium: Active Traffic Pattern Prediction for Boosting P2P Collaboration, *Proceedings of the 2009 IEEE 18th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2009, IEEE Computer Society

Horovitz, S. & Dolev, D. (2009b). Maxtream: Stabilizing P2P Streaming by Active Prediction of Behavior Patterns, *Proceedings of the 2009 Third International Conference on Multimedia and Ubiquitous Engineering*, pp.546-553, 2009, IEEE Computer Society

Horvath, A.; Telek, M. ; Rossi, D. ; Veglia, P. ; Ciullo, D.; Garcia, M. A. ; Leonardi, E. & Mellia, M. (2008). Dissecting PPLive, Sopcast, TVants, *Technical report, Politecnico di Torino*

IPoque Internet Study – The Impact of P2P File Sharing (2007), *http://www.ipoque.com/userfil es/file/internet_study_2007.pdf*

Izhak-Ratzin, R. (2009), Collaboration in BitTorrent Systems, *Networking 2009: 8th International IFIP-TC 6 Networking Conference*

NewTeeVee report - Joost bandwidth problems (2008), *http://newteevee.com/2008/03/20/wher e-to-watch-marchmadness/*

Piatek, M.; Isdal, T.; Krishnamurthy, A. & Anderson, T. (2008), One hop reputations for peer to peer file sharing workloads, NSDI'08: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*

Pouwelse, J.; Garbacki, P.; Wang, J.; Bakker, A.; Yang, J.; Iosup, A.; Epema, D.; Reinders, M.; van Steen, M. & Sips, H. (2006). Tribler: A social-based peer-to-peer system. *IPTPS06*

Rejaie, R. & Ortega, A. (2003). Pals: Peer-to-peer adaptive layered streaming. *NOSSDAV'03 Monterey, CA*

Sandvide 2008 Global Broadband Phenomena (2008), *http://www.sandvine.com/general/documents/2008-global-broadband -phenomena - executive summary.pdf*

Schlosser, D.; Hossfeld, T. & Tutschku, K. (2006). 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, 2006*

Souza, L.; Cores, F.; Yang, X. & Ripoll, A. (2007), DynaPeer: A Dynamic Peer-to-Peer Based Delivery Scheme for VoD Systems. *Euro-Par 2007 Parallel Processing, ISBN 978-3-540-74465-8, 2007*

TechCrunch report - Joost abandons p2p (2008), *http://www.techcrunch.com/2008/12/17/joost-just-gives-upon-p2p/*

Vapnik, V. N. (1995), The nature of statistical learning theory. *Springer-Verlag New York, Inc., 1995.*

VentureBeat report - Joost playback problems (2008), *http://venturebeat.com/2008/11/2 8/joost-is-loosed-onthe-iphone-if-only-it-worked/*

Vu, L. ; Gupta, I. ; Liang, J. & Nahrstedt, K. (2006), Mapping the PPLive network: Studying the impacts of media streaming on p2p overlays, *Technical report, August 2006*

Wang, J. (2008). Robust video transmission over lossy channels and efficient video distribution over peer-to-peer networks, *Technical Report, University of California at Berkeley, 2008*

Wang, J.; Yeo, C.; Prabhakaran, V.& Ramchandran, K. (2007). On the role of helpers in peer-to-peer file download systems: Design, analysis and simulation. *IPTPS07*

Wang, J. & Ramchandran, K. (2008), Enhancing peer-to-peer live multicast quality using helpers, *Image Processing, 2008, ICIP 2008*

Wong, J. (2004), Enhancing collaborative content delivery with helpers. *Master's thesis, Univ of British Columbia, 2004*

Parallel and Distributed Computing

Edited by Alberto Ros

ISBN 978-953-307-057-5

Hard cover, 290 pages

Publisher InTech

Published online 01, January, 2010

Published in print edition January, 2010

The 14 chapters presented in this book cover a wide variety of representative works ranging from hardware design to application development. Particularly, the topics that are addressed are programmable and reconfigurable devices and systems, dependability of GPUs (General Purpose Units), network topologies, cache coherence protocols, resource allocation, scheduling algorithms, peertopeer networks, largescale network simulation, and parallel routines and algorithms. In this way, the articles included in this book constitute an excellent reference for engineers and researchers who have particular interests in each of these topics in parallel and distributed computing.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Shay Horovitz and Danny Dolev (2010). On the Role of Helper Peers in P2P Networks, Parallel and Distributed Computing, Alberto Ros (Ed.), ISBN: 978-953-307-057-5, InTech, Available from: http://www.intechopen.com/books/parallel-and-distributed-computing/on-the-role-of-helper-peers-in-p2p-networks

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen