periodica polytechnica

Electrical Engineering
55/3-4 (2011) 111–117
doi: 10.3311/pp.ee.2011-3-4.03
web: http://www.pp.bme.hu/ee

RESEARCH ARTICLE

Received 2012-07-11

Small degree BitTorrent

Miklós Kasza / *Vilmos* Bilicki / *Márk* Jelasity

Abstract

It is well-known that the BitTorrent file sharing protocol is responsible for a significant portion of the Internet traffic. A large amount of work has been devoted to reducing the footprint of the protocol in terms of the amount of traffic, however, its flow level footprint has not been studied in depth. We argue in this paper that the large amount of flows that a BitTorrent client maintains will not scale over a certain point. To solve this problem, we first examine the flow structure through realistic simulations. We find that only a few TCP connections are used frequently for data transfer, while most of the connections are used mostly for signaling. This makes it possible to separate the data and signaling paths. We propose that, as the signaling traffic provides little overhead, it should be transferred on a separate dedicated small degree overlay while the data traffic should utilize temporal TCP sockets active only during the data transfer. Through simulation we show that this separation has no significant effect on the performance of the BitTorrent protocol while we can drastically reduce the number of actual flows.

Keywords

component · BitTorrent · small degree overlay

Acknowledgement

M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

Miklós Kasza
Vilmos Bilicki
University of Szeged, 6720 Szeged, Aradi vértanúk tere 1., Hungary

Márk Jelasity
University of Szeged, Hungarian Academy of Sciences, 6720 Szeged, Aradi vértanúk tere 1., Hungary

1 Introduction

BitTorrent traffic is responsible for a large portion of the overall Internet traffic. Accordingly, a lot of attention has been devoted to optimizing BitTorrent clients, for example, to make them ISP friendly, via localizing data traffic [1]. However, these works have focused on traffic volume almost exclusively. At the same time, with the advent of the intelligent network [2] we are witnessing the development of network services where the load is proportional to the number of flows and not traffic volume. In addition, at lower layers of the network, we also see sensitivity to the number of flows, for example, in the TCP layer of wireless networks.

This work is devoted to the study of the flow level behavior of BitTorrent, through realistic simulations. Based on the observations of these simulations, we propose an approach to drastically reduce the number of flows in a BitTorrent swarm. We also demonstrate that our approach has an affordable performance penalty.

1.1 Novelty and Contribution

1.1.1 The number of the flows is important from the point of view of single nodes as well as the whole BitTorrent ecosystem

We discuss possible scalability issues in the different layers of the network infrastructure that are caused by the high number of flows (as opposed to a high volume of traffic). We conclude that the access, distribution and core layers are affected as well.

1.1.2 Stability of data transfer flows

We are not aware of any studies focusing on the stability and the dynamics of the BitTorrent data transfer graph. Empirical studies and mathematical models are known of the clustering of clients based on bandwidth [3], however, we study the case when the bandwidths can be equal, and we focus on the stability and dynamical properties of flows. We show that, due to the Tit-For-Tat algorithm (which takes into account the history of the communication) and due to the efficiency of the random first algorithm, flows are remarkably stable, which is a very important property from the point of view of our approach to implement-

ing a small degree BitTorrent swarm.

1.1.3 Extended model for the BitTorrent protocol

To perform realistic simulations, we reverse engineered the most significant parts of the Vuse [4] BitTorrent implementation. There are several interesting findings we have not seen in applied models (e.g.: the implementation of optimistic peer selection which is not uniform random, but instead it is weighted based on the history of the bilateral communication).

1.1.4 Separated data and signalling paths

We propose to apply a separate mechanism for signaling and data transfer. With the help of realistic simulations we show that this could be done without a serious performance penalty.

1.2 Outline

The outline of the paper is as follows. In Section 2 we discuss necessary background concepts along with the motivation for the scalability problem that we tackle in this paper. Section 3 discusses our approach to model a BitTorrent swarm. This is important as the implemented simulation (based on the Vuse BitTorrent client) is itself a contribution of this work. Section 4 contains a recap of our previous work in the area of small degree DHTs, and outlines the applicability of this concept for implementing a small degree BitTorrent overlay. It also discusses our approach to modeling this overlay. Section 5 presents experimental results with our model. Finally, Section 6 concludes the paper.

2 Backgroung and the scalability problem

2.1 The BitTorrent overlay

The strength of the BitTorrent protocol lies in its efficiency of entropy maximization and preventing free riders from obtaining significant resources. The latter goal is achieved with the help of the Tit-For-Tat mechanism while the former goal is achieved with the help of the rarest first chunk selection algorithm.

The peers participating in a swarm maintain TCP connections with each other. These connections are used for data exchange and signaling. The properties of the overlay defined by these connections has been extensively studied. For example, the authors of [5] found that the number of connections has a significant effect on the diameter of the overlay. The overall number of connections a node can maintain determines the number of its outgoing and incoming connections. The larger the number of outgoing connections, the smaller the diameter of the overlay becomes, which is desirable. Besides, if the number of incoming connections is limited in practice, the nodes form clusters based on the time they joined, which is undersirable. Overall, from all points of view, an overlay of a large degree is desirable.

However, as the connections in BitTorrent are all real TCP connections (while in eDonkey, for example, there are logical connections based on UDP signaling) the upper bound on the overall number of connections is based on the efficiency of TCP sharing a given link with a given bandwidth [6]. Based on the default settings 80 connections are allowed for each node: 40 connections initiated by the node and 40 connections initiated by the peer nodes participating in the swarm [6].

2.1.1 The impact of the number of the concurent connections on the infrastructure

The impact of the number of the concurrent TCP or UDP connections on the infrastructure before or after the POP (Point of Presence) has not been widely studied. Here we briefly summarize the scalability issues related to the number of flows, as opposed to traffic volume (both for a single BitTorrent client and for the whole BitTorrent ecosystem).

In related work the following issues are known regarding the number of *parallel connections*:

- Most of the home routers or xDSL/Cable modems can provide NAT. Many older or cheaper devices can only maintain up to 250 parallel connections [7, 8].
- The overhead (ACK) of the TCP protocol that consumes the uplink capacity should also be considered. The best practice is to limit the number of per torrent connections to a value of 25 to 100 connections (based on the upload speed), as suggested in [9].

Another issue is related to *intelligent networking*. The integrated network infrastructure where the crosscutting issues of the QoS, Security, Mobility, Efficiency, etc, should be solved are going to be more and more "Intelligent". In contrast to past best practices where the core was dumb and the edge was intelligent there are now numerous cases where even the core should have some kind of intelligence [2].

In the following we would like to give an overview of lower level services where the number of concurrent flows could yield scalability issues. In the access layer the wireless connections are gaining significant market share. It is not exceptional to run a BitTorrent protocol over a 3G or WiFi connection. The following solutions (now or in the future) may have scalability issues according to the number of concurrent flows:

- The packet loss or the delayed packets are interpreted by TCP as congestion, decreasing the sending window as a result, and in this way decreasing the throughput as well. However, in the case of wireless media, packet loss often happens due to the medium and not because of a buffer overflow. In this case TCP misinterprets packet loss or delay events. There are techniques for overcoming this issue in different network layers but most of these techniques should threat each TCP flow separately. For example, TCP snooping at the NODE-B (Base station) [10], the so called Ack-Regulator, maintains a queue for each flow [11] at the RNC (3G - Radio Network Controller).
- It is common that in the 3G the user equipment receives a private IP address which is then NAT-ed mostly at the gateway

GPRS support node (GGSN) [12]. As one GGSN could aggregate all the IP traffic, the high number of flows could cause scalability problems.

Together with the integrated services the network infrastructure becomes an integral part of everyday life. *Reliability and security* are becoming the main aspects of network management. The classical view of a dumb core and a smart edge is now challenged by national and international regulations where the ISPs are becoming responsible for the security of the their network. Network monitoring and especially flow level monitoring are the most important information sources for security and traffic engineering.

However, the scalability of the current monitoring framework has been studied only in a few articles [13, 14]. It is clear that with the increasing number of flows the resource consumption of these solutions is also increasing. One could argue that packet sampling could help, but weaknesses of the packet sampling approach have been pointed out [15]. The scalability of the monitoring infrastructure of the ISP is one aspect which is not considered when talking about the number of concurrent flows a BitTorrent client maintains, but it should also be considered when someone is considering the whole BitTorrent ecosystem and ISP friendly P2P solutions.

To sum up, we have seen that the number of paralell TCP/UDP connections a node maintains has an important impact and barriers not only in the access layer but also in the distribution and core layer too. We can conclude that the small degree of the nodes is desirable both for the end users and the ISPs.

2.1.2 Stability and Clustering

It is well known that due to the Tit-For-Tat algorithm the links among the nodes are not utilized equally. The authors of [16] and [17] studied the node centric behavior of the BitTorrent protocol with the help of measurements conducted on real torrents. They found that each peer selects a small subset of peers for data exchange. The remaining connections are mainly used only for keep alive and signaling activity. In article [18] the authors conducted measurements and also found that BitTorrent nodes form clusters based on the common upload speed. We should remark here that this clustering depends strongly on the so called peer availability but in most cases peer availability is high [16–18].

3 Modeling the BitTorrent swarm

In order to gain useful insights into the flow level behavior of BitTorrent, our main focus in this work, it is important to work with a realistic model of a BitTorrent swarm.

As a result of the numerous system and node level measurement projects there are several commonly accepted parameters for the characterization of a swarm which could help the researchers to build better models for simulation based studies. These parameters are trying mainly to capture the aggregate end user behavior at the system level. The node level model of the BitTorrent protocol itself varies from study to study. Here we describe the model we constructed in order to simulate the swarm. The details of the protocol are based on the reverse engineered code of the Vuse BitTorrent client.

3.1 Peer arrival and session length

In the current study we applied the simplest model where all the nodes start the download activity at the start of the simulation and they remain active during the whole simulation.

3.2 Neighbor selection

For each swarm member the primary source of information about other swarm members is the tracker of the torrent. Nodes regularly communicate with the tracker to keep a number of peer connections. In our model we used a widely accepted default of the total of 80 simultaneous peer connections in downloading mode out of which 40 connections are activated locally and 40 are initiated remotely.

3.3 Choking algorithm

For every BitTorrent implementation the choking algorithm is a crucial point. In our model we followed the specification [6] but refined it based on Vuze.

Initially, for each peer all the other peers are in choked state and unchoking decisions are re-evaluated periodically (by default every 10 seconds both in the Omnet++ model and in Vuze), but this period should not be too short in order to avoid fibrillation, the case when a peer gets quickly choked and unchoked repeatedly.

Usually there is a hard limit for the number of simultaneously unchoked channels. The Omnet++ model [9] and Vuze implementation [6] both use 5 as an upper limit, and one slot is reserved for optimistic decisions.

Peers usually use historic information (upload and download rates, number of downloaded bytes, etc.) for deciding which peers to unchoke, but selecting the right set of such peers is a challenging task. The decisions should reciprocate to peers who let the node download. This is achieved by selecting interested peers based on their upload rates (peers with best upload rates get unchoked). Specification [2] enables the selection of non-interested nodes to get unchoked, however, Vuze does not select such nodes and neither our implementation does.

Our implementation follows Vuze's decision algorithm [6]. The algorithm used while downloading is different from the one applied during seeding, however it is always a goal to maximize the number of simultaneously unchoked peers up to the limit.

In downloading mode the algorithm is the following. First we filter out very slow peers and sort the remaining ones by their upload rates. Upload rates are computed by averaging the data upload rate in last 20 seconds. Afterwards, we collect the best of the interested peers in the sorted collection, as these peers are the first candidates for unchoking. If the number of such peers

is less than the desired 4 we collect some peers we have already communicated with and uploaded a fair amount of data to us (their download/upload ratio does not exceed 3). These peers are sorted by the total amount of data they've sent. The remaining slots, up to 5, are filled with optimistically selected peers. This choking decision process is done in every 10 seconds. During these 10 second periods only optimistic peers can replace disconnected peers. In every 30 seconds at least one optimistically selected peer gets choked and another peer gets unchoked optimistically.

In seeding mode the algorithm uses other criteria for unchoking. First, peers are ranked by their data receive rate averaged in the last 5 seconds and the total amount of bytes uploaded to them. Peers downloading faster but having downloaded smaller amount of date are preferred. Optimistically selected peers are unchoked similarly to downloading mode.

Optimistic peer selection means that random peers are selected for unchoking even if they are not interesting or do not have good upload/download characteristics. This mechanism introduces the randomness into the system that is needed to give peers a chance to catch up with the swarm. Note that in downloading mode Vuze and our model takes the reciprocation criterion into consideration and thus random decision are weighted by data upload/download history of the peers based on the following method:

- collect unchokable peers to a list L and rate them based on the total number of bytes uploaded to (U[p]) and downloaded from (D[p]) them, store the rate values to an array R:

```
for each choked peer p:
        if unchokable(p):
                add p to L
                R[p] := U[p] - D[p]
```

- sort elements of L by their rate maintaining descending order (thus the elements at the front of the list are those that downloaded the most and uploaded the least)
- generate a random integer x in the interval [0, size(L))
- in order to select at most N peers optimistically, repeat the following steps at most minN, size(L) times:
  - use the following formula to calculate index of each optimistically selected peer (not that the formula prefers elements at the end of the list):
    `index := 0.8 + 0.2 * x`$^{-1}$
    `selectedPeer := L[index]`
  - remove the selected peer from L

3.4 Piece selection strategy

Nodes can freely choose which pieces they want to get from a neighbor, however different piece selection strategies exist. In the Omnet++ model *rarest first* and *random first* strategies are implemented while Vuze applies a more sophisticated algorithm and uses piece prioritization based on different factors. Our implementation uses the rarest first method.

The algorithm does the following for any specific peer *p* that unchoked the node. First it creates a list of pieces available from p and for each available piece it counts how many of the neighboring peers can provide the same piece. The available pieces are sorted into buckets based on the number of peers that can provide them. The algorithm randomly selects a piece from the first non-empty bucket that has the smallest occurrence value.

4 Small degree BitTorrent

So far we have argued that for BitTorrent a large degree overlay is desirable for optimal performance, however, for the network infrastructure a very small degree overlay is desirable. These conflicting goals have to be resolved in a BitTorrent client. In this section we briefly outline a solution for this problem.

4.1 Signalling and data channels

First of all, to reduce the effective communication partners of BitTorrent clients, we need to understand the difference between signaling traffic and data traffic. Signaling involves infrequent communication of small packets, while data transfer involves frequent transmissions of large volumes of data. From the point of view of the number of network flows however, they are equal.

Our main insight here is that if we could compress signaling traffic into a clever information dissemination layer, while keeping the data traffic essentially unchanged, then if there were only relatively few data flows, then the overall number of flows could be reduced.

We will prove in the next sections that there are indeed only a few data flows. Compressing the signaling traffic can be implemented, for example, through a small constant degree distributed hash table, that routes this traffic through a small number of connections [19]. This DHT can even perform optimizations such as implementing an efficient multicast service [20], providing further savings in traffic.

It is important to emphasize that the number of *virtual* connections remains the same: the DHT acts as a middleware that can even hide the fact that the actual neighbors the client communicates with are from a small constant set. In fact, the DHT based signaling approach makes it possible to increase the number of virtual neighbors well beyond current limits.

4.2 Modelling the tunnels

In the following, instead of implementing a full-blown DHT layer below our simulated BitTorrent client, we opted for a relatively simple solution that nevertheless offers a worst case approximation of the realistic scenario: we model the DHT layer via adding a large delay, in the order of 3 seconds, to signaling traffic, to capture the overhead of the indirection through the routing layer of the DHT.

5 Experimental Evaluation

In order to model large swarms we implemented a BitTorrent client in the cycle based model of Peersim [21]. As was shown in [22], flow level models could provide an acceptable approximation of the real protocol. The protocol itself was based on the codebase of the Vuse implementation. We selected the Vuse implementation because it is one of the most popular BitTorrent implementations (it has about 25% market share) and it is Java based as well, so the transformation of its logic for Peersim (a Java based simulator) was relatively simple.

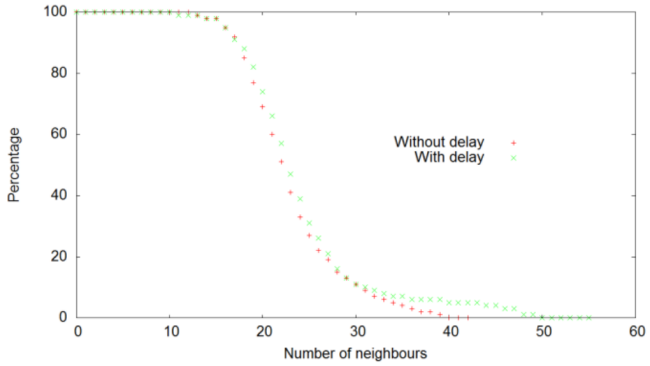

Fig. 1. Distribution of the number of neighbours

The important parts of the code (peer selection, peer maintenance, piece selection, piece maintenance, etc) were reverse engineered with all the details (magic numbers, etc) transformed/adopted to our simplified data structure. In this aspect the model is unique as the implementation and the behavior is the same as that of the real Vuse client.

Network traffic was modeled as the solution of a linear constrained optimization problem. The communication flows were based on the modeled piece message exchanges. We defined two groups of inequalities:

- The amount of the uploaded and the downloaded data should not exceed the available upload and download bandwidth
- The size of the piece should not be exceeded. In other words only the remaining data should be transferred on a given connection.

We defined the cost function of the constrained optimization problem as follows:

- The data transformed in a given interval should be maximal
- The flows should share the available resources equally if possible.

After each cycle the BitTorrent piece messages to be transferred were determined, and the size of the data actually transferred in these messages was estimated with the help of the solution to the constrained optimization problem.

5.1 Simulation setup

In our setup one cycle corresponds to one second. The upload bandwidth of the nodes was set to 1024 MBit/s while the download bandwidth was set to 4048 MBit/s. The size of a piece was set to 1024 KBytes while the file to be downloaded was set to 100 MBytes. The number of nodes taking part in the swarm was set to 1000 during the lifetime of the simulation. One of the nodes was the seeder operating in seeding mode from the start. The tracker gave 20 random nodes for each request. The timing of the choke/unchoke was taken from Vuse as described previously. The effect of the churn on the dedicated Kademina based signaling overlay channels was modeled as delays with linear random distribution up to 3 seconds. In other words each signaling message was randomly delayed to model the routing time through the DHT. All the simulations were run for 2000 cycles (2000 seconds)

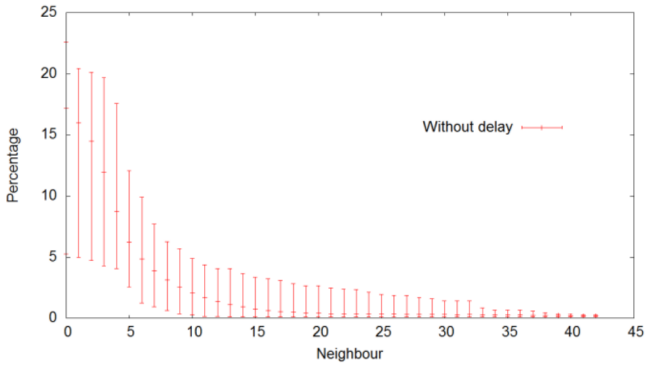

Fig. 2. Data exchange stability without delay

5.2 Results: properties of the overlay

The first simulations were conducted in order to study the properties of the download graph of the BitTorrent ecosystem. We performed the simulations with and without delays, where delays model the underlying DHT layer for handling the signaling traffic, as described previously. The results are shown in Figure 1. Figure 2. and Figure 3.

During the simulation we collected the node pairs involved in data upload/download in each cycle. For each node, we calculated the amount of traffic that was transferred through each of these pairwise connections.

Figure 1. shows the distribution of the number of data exchange neighbors with and without delay. We can see that most of the nodes had less than 30 data exchange partners (that is, peers with a non-zero volume of exchanged data) during the experiment. The effect of the delay is visible but it is not significant.

The number of neighbors that were used for data exchange does not completely reflect the stability of the network. In fact, the number of neighbors that were used for a significant amount of data exchange is much less.

In order to be able to study the swarm level stability of

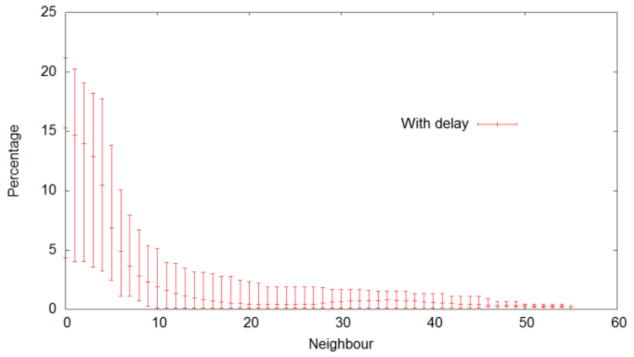

Fig. 3. Data exchange stability with delay

the connections we aggregated node level traffic statistics into swarm level statistics in the following way: we took the list of data exchange peers ordered by the percentage of the number of active communication cycles vs. total communication cycles in descending order and calculated the swarm level minimum, maximum, and average for each rank. These statistics are shown on the Figure 2. and Figure 3.

On the horizontal axis the rank of the neighbor is shown ordered by the percentage of the data exchange cycles in decaying order. On the vertical axis percentage values are shown. The minimum, maximum and average values are shown with the help of error bars. For example, in Figure 2. , the error bar in position 2 indicates that for the nodes the second most stable peer accounted for 17% of the data traffic on average, while for the whole swarm the biggest amount for this position was 21% and the smallest was 5%.

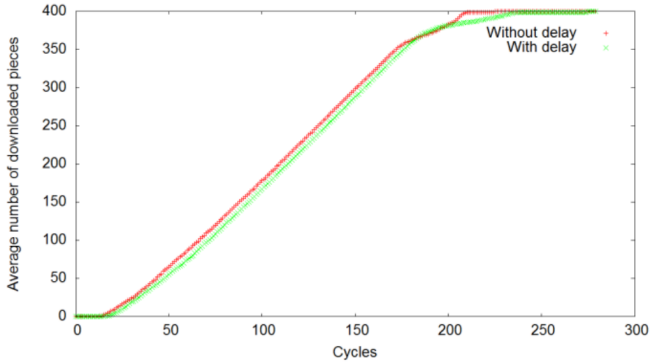

Fig. 4. Performance with and without delay

We can conclude that during the simulation the neighbors were quite stable as the average activity of the first 6 partners took more than 69% of the total communication on average. There is no significant difference between the results of the simulations with delay and without delay.

5.3 Results: performance effects of the separated signalling channel

One important question is the effect of the delay on the performance of the torrent. We have measured the average number of the pieces on the nodes, shown in Figure 4. The figure presents the number of pieces as a function of time. We can conclude that there is a performance penalty is not significant the difference is in the order of several percentages. On the other hand, we note that routing signaling traffic through a small degree DHT (the case modeled by the delay) makes it possible to increase the number of signaling connections very significantly. This higher number of potential connections could help to decrease this penalty; although we have not examined this scenario in this paper.

6 Conclusions

In this paper we have formulated and motivated a problem that is related to potential scalability issues of BitTorrent swarms: we argued that the large number of flows, irrespective of traffic volume, represents a scalability bottleneck, because the last generation of intelligent network monitoring and filtering applications are sensitive to the number of flows, as well as the lower level TCP implementations, especially in wireless media.

To solve this problem, we have implemented a realistic BitTorrent simulation in PeerSim, and used the simulations to demonstrate two results. The first result is that only very few connections are used to transfer large amounts of data, most of the connections are used for signaling. This makes it possible to route signaling traffic through a small degree DHT, which drastically reduces the number of flows.

The second result was that this solution does not spoil the stability of the connections that are used for data transfer, while it does not significantly reduce download the throughput either.

References

1 **Slot M, Costa P, Pierre G, Rai V**, *Zero-Day Reconciliation of BitTorrent Users with Their ISPs*, Euro-Par 2009 Parallel Processing (2009).

2 **Kephart J., Chess D**, *The vision of autonomic computing*, Computer **36** (2003), 41–50.

3 **Meulpolder M, Pouwelse J A, Epema D H J, Sips H J**, *Modeling and analysis of bandwidth-inhomogeneous swarms in BitTorrent*, Ninth IEEE International Conference on Peer-to-Peer Computing (P2P'09), posted on 2009, 232–241, DOI 10.1109/P2P.2009.5284523, (to appear in print).

4 *Vuze: The most powerful bittorrent client in the world*, `http://www.vuze.com/`.

5 **Al-Hamra A, Liogkas N, Legout A, Barakat C**, *Swarming Overlay Construction Strategies*, Proceedings of 18th International Conference on Computer Communications and Networks, posted on 2009, 1–6, DOI 10.1109/ICCCN.2009.5235297, (to appear in print).

6 *BitTorrentSpecification, TheoryOrg*, `http://wiki.theory.org/BitTorrentSpecification#Tracker_HTTP.2FHTTPS_Protocol`.

7 *Vuze FAQ / What do I do if my network connection keeps dying?*, `http://faq.vuze.com/?View=entry&EntryID=137`.

8 *How Many Connections Can A Wireless Router Handle? | PCMech*, `http://www.pcmech.com/article/how-many-connections-can-a-wireless-router-handle/`.

9 *Good settings - VuzeWiki*, `http://wiki.vuze.com/w/Good_settings`.

10 **Balakrishnan H, Seshan S, Katz R. H**, *Improving reliable transport and handoff performance in cellular wireless networks*, Wireless Networks **1** (1995), 469–481, DOI 10.1007/BF01985757.

11 **Chan M C, Ramjee R**, *TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation*, Wireless Networks **11** (Jan. 2005), 81-97, DOI 10.1007/s11276-004-4748-7.

12 **Bannister J, Mather P. M, Coope S**, *Convergence technologies for 3G networks: IP, UMTS, EGPRS and ATM*, John Wiley & Sons Inc, 2004.

13 **Cisco Systems**, *NetFlow Performance Analysis*, 2007.

14 **Kadlicsek J., Pásztor G.**, *Netfilter Performance Testing*.

15 **Haddadi H., Landa R, Moore A.W., Bhatti S, Rio M, Che X**, *Revisiting the issues on netflow sample and export performance*, Third International Conference on Communications and Networking in China (2008), 442-446.

16 **Legout A, Urvoy-Keller G, Michiardi P**, *Understanding BitTorrent: An Experimental Perspective* (2005).

17 **Legout A, Liogkas N, Kohler E, Zhang L**, *Clustering and sharing incentives in bittorrent systems*, Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (2007), 312-.

18 **Legout A, Urvoy-Keller G, Michiardi P**, *Rarest first and choke algorithms are enough*, Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (2006), 203–216.

19 **Jelasity M, Bilicki V**, *Scalable P2P Overlays of Very Small Constant Degree: An Emerging Security Threat*, Stabilization, Safety, and Security of Distributed Systems, 399–412.

20 **Castro M, Druschel P, Kermarrec A, Nandi A, Rowstron A, Singh A.**, *SplitStream*, Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03 (2003), 298-.

21 **Montresor A, Jelasity M**, *PeerSim: A scalable P2P simulator*, IEEE Ninth International Conference on Peer-to-Peer Computing (2009), 99-100.

22 **Eger K, Hossfeld K, Binzenhöfer A, Kunzmann G**, *Efficient simulation of large-scale p2p networks*, Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks - UPGRADE '07 (2007), 9-.