Attacks and exploits targeting BitTorrent and other P2P file sharing networks

Andreas Hegenberg
Betreuer: Benedikt Elser
Seminar Future Internet WS 09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: andreas.hegenberg@in.tum.de

***Abstract*— This proceeding deals with various attacks and exploitations P2P networks have to cope with. In the last few years BitTorrent became famous for spreading big amounts of data to many people in a short time. BitTorrent will therefore be the main focus of this work. This proceeding serves basically as overview about different types of attacks. It shows attacks that are able to prevent files from being downloaded or at least delay the downloading process enormously. For some of the attacks countermeasures are shown. Furthermore the proceeding discusses so-called free riding which allows a peer to download without reciprocating. Understanding how the different attacks and exploits work is a important step to design more resilient protocols in the future.**

***Keywords*— P2P, attacks, BitTorrent,free riding**

I. Introduction

The amount of fast Internet connections increases steadily and so does the need for cheap but yet fast content distribution. Peer to Peer (P2P) file sharing systems are very popular for quickly spreading high amounts of data to many people without much effort or cost. Unfortunately there are several groups which are interested in attacking or exploiting those networks for different reasons. For example the music, film and television industries already (successfully) attacked various P2P networks because they hoped to stop the illegal distribution of their assets. But also regular users can cause problems if they try to download without reciprocating in any form to the community, this is called free riding and could cause the entire network to slow down extremely.
This proceeding will deal with the question how resilient today's overlay P2P systems are against a variety of possible attacks and/or exploitations. Therefore I will describe some potential types of attacks and their impact to harm P2P systems. Because BitTorrent has evolved to the currently most used P2P network and because it is available as open source this work will concentrate on it.
The first chapter will give you a very basic introduction to the BitTorrent protocol. The second chapter will be about different types of attacks that prevent files from being downloaded. The third chapter will then deal with the incentives system in BitTorrent and how to trick it. Finally followed by a short discussion about future work and other P2P file sharing protocols.

II. BitTorrent

BitTorrent is an open source peer to peer file-sharing protocol which became very popular during the last few years. Today BitTorrent is responsible for a huge amount of traffic on the internet. The 'Internet Study 2008/2009' released by the german company Ipoque at the beginning of the year estimated the traffic produced by BitTorrent users at 27-55% of all internet traffic, depending on the geographical location. Meanwhile BitTorrent serves as blueprint for many applications, e.g. for streaming software. Even the media industries checks if BitTorrent could be used for the commercial distribution of movies, music, tv-shows etc.. [1]

A. BitTorrent Characterization

In BitTorrent a file is divided into many pieces (usually each pieces size is 256KB) those pieces are subdivided into blocks (usually 16KB). The information how a file is split gets saved in a metafile, which also holds SHA1 hashes for every piece (but not for the blocks) and the URL of a tracker server. The tracker is a centralized entity which is responsible for managing the peers that belong to a specific file and for assisting their communication. Therefore the tracker maintains a list of currently active peers and delivers a random subset of these to clients, upon request. (see figure 4) [2]. Usually trackers are also the platforms where users can download the metafiles via standard HTTP. As you see BitTorrent in its original specification is not pure P2P but has some client-server attributes too.
Peers (clients) that want to download a single file or already got that file are grouped into swarms - as long as their client is running. A swarm usually consists of seeders and leechers whereby a seeder is an user who already possesses all the pieces of a file.
A leecher uploads blocks he already got to other leechers that request those blocks from him. The other leechers are able to determine which pieces he has based on bit-field messages they exchange [3]. Peers chose the pieces they request using a local rarest first policy [4], this means they chose those pieces which are least replicated among the peers they are connected to. This shall ensure a balanced piece availability.

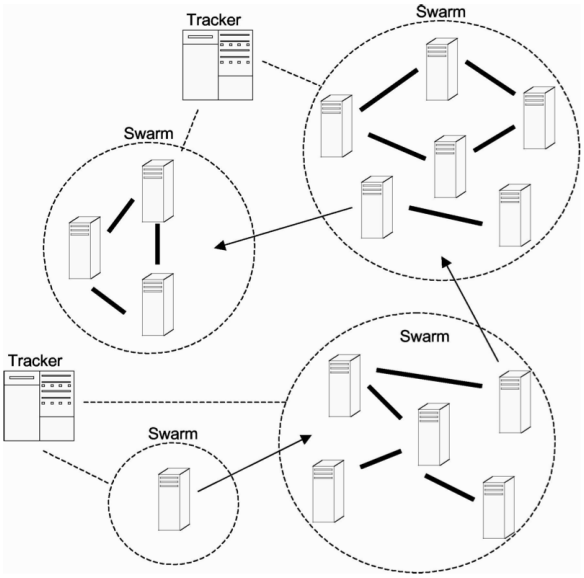

Fig. 1. Portion of a BitTorrent system, arrows show how users could jump from one swarm to another or even be in both swarms at the same time. [5]

The decision which other leechers will get one of his n upload slots (usually there are 4-5 slots) is made using an incentive mechanism. [4]. The standard mechanism works periodically, one period typically takes 10 seconds. In a period the client sorts the list of peers he has currently downloaded from by the speed they offered to him. In the next period he unchokes a specific number (n-1) of peers that achieved highest rates and chokes the other peers. Unchoking means, that the selected peer gets one of the n upload slots and choking means he will lose his upload slot (but stay connected) see figure 2.

This behaviour is called Óbandwidth firstÓ. Additionally he selects another random peer for an optimistic unchoke every third period. [6] This makes finding new, maybe better peers possible and also allows new clients to gather some data so they can become uploaders fast. Through this tit-for-tat system a fair trading shall be ensured. Later in this work we will see if it works well.

If a leecher has all parts of a file he becomes a seeder. Seeders can not consider any download speed for unchoking (because they download anymore) thus the upload speed is watched. The peers to which a seed can upload fastest are then chosen in a round robin fashion. [3]

Because BitTorrent is open source, the protocol is implemented in many slightly different ways from various clients. A closer look will show that some of those implementations are considerably more prone to specific attacks than others.

If you need a more specific description of the BitTorrent protocol take a closer look on the Bittorrent protocol specification. If needed we will describe some specific parts of the protocol in the course of this work.

III. Attacks on BitTorrent

The authors of [7] [8] [1] [4], describe various attacks on BitTorrent swarms or parts of a swarm which could make it impossible to download a file or slow down the whole process enormously. This chapter addresses various attacks of this type.

A. Attacks against Seeders

Targeting seeds is a very natural approach because without a seed it clearly becomes hard to complete the download for any leecher. However the attack has to take place in an early stage of distribution, if all the parts are already spread to leechers it is often too late. [7] [8] In this section we will discuss different seed attacks. In almost every seed attack identifying the initial seeders as fast as possible is a necessary precondition to render the attack successful. Unfortunately this is not too hard as [7] shows. Because most big torrent web sites give their files an incrementing index number new files can easily be watched. Also some trackers offer public RSS feeds of all new files.

1) Early stage DoS attacks on seeders : Early stage Denial of Service attacks which aim on cutting off the initial seeder from the network seem to be an effective way to take down a whole swarm. The results of [8] confirm this. Their measurements show that if a seeder is cut off from the network due to some sort of DoS attack while the average download progress is below 20% almost always the leechers can not complete their download . They also show that it is hard to predict what happens if the average download ratio is between 20% and 60%. But it is obvious that the probability of completing the download increases with the average download ratio. Above 60% the download could always be completed for every leecher.

In [8] is not described how the DoS attack is executed. The simplest approach would be to launch massive requests at the seeder. However flooding DoS attacks need quite a lot resources especially if they are directed to many popular torrent files or if the seeder is a host with a fast internet connection. So instead of running this simple kind of DoS attack it could be more efficient to exploit specific characteristics of the BitTorrent protocol [1] Unfortunately it is hard to circumvent those attacks. Newer BitTorrent clients support a feature called super seeding [9] which intends to spread a file faster. As a sideline this feature hides that the client is a seeder. Maybe this is a good start for hampering seeder attacks, but there definitely is much more room for improvements and further investigations.

2) Bandwidth attacks on seeders: The authors of [7] describes two attacks against seeders, the first we will discuss is called bandwidth attack. The approach is quite simple, the attacker attempts to consume the majority of the seeds upload bandwidth [7].

For understanding how seeders assign their upload slots to leechers the authors examine two actual client implementations of seeding algorithms, the one of Azureus version 3.1.1.0 (meanwhile called Vuze) and the one of BitTornado (unknown version).
According to [7] Azureus uses a variation of the original bandwidth first seeding algorithm. This algorithm prefers users who have a high download speed and at the same time low amount of data downloaded from the seed. The other client, BitTornado uses a pure bandwidth first algorithm when seeding [7].
Both implementations perform optimistic unchokes.
Two cases of bandwidth allocation between seeders, leechers and attackers were considered during the experiments. In the first case the seeders upload bandwidth per upload slot is lower than the leechers download bandwidth and this is lower than the attackers download bandwidth. This means both, the leechers and the attackers can only download at the same speed (limited by the seeder) and thus the attackers have no real advantage by their higher bandwidth.
The second case where seeders achieve higher upload speed per slot than leechers can download but lower than attackers can download seems to be more harmful if a bandwidth first algorithm is used.
As the measurements in [7] show Azureus' implementation ensures a high resilience against bandwidth attacks in every case because even if the attackers download very fast they will soon have a high amount of data downloaded from the seed and thus get choked. During their experiments the delay ratio was always less than 3 compared to the same download without attackers.
Surprisingly BitTornado performed only slightly worse. This is because in the second bandwidth allocation case the upload bandwidth of the seeder is higher. Thus even if the attackers obtain all regular upload slots, the optimistic unchoke from time to time makes sure a friendly peer downloads at high rate.
Unfortunately you will see that the seeding algorithm used by Azureus has a major disadvantage if you read on.

3) Connection/Eclipse attacks on seeders: Every BitTorrent client has a maximal number of connection slots that can be filled by other peers (usually about 50). The so-called connection attack, also known as eclipse attack (the second attack described in [7]) aims on filling the vast majority of those slots with malicious peers, so that no friendly peers can connect anymore. This shall be done using as low bandwidth as possible.
As the investigations in [7] show Azureus is higly vulnerable to this attack, in the authors experiments. BitTornado performs better but also is affected. Their attack environment consists of a single seed sharing a file, 30 leechers, a single tracker and a number of attack peers [7]. They try to simulate a flash crowd effect by starting 5 leechers first, then after those are connected to the seed all the attackers are launched, and finally the rest of the leechers.

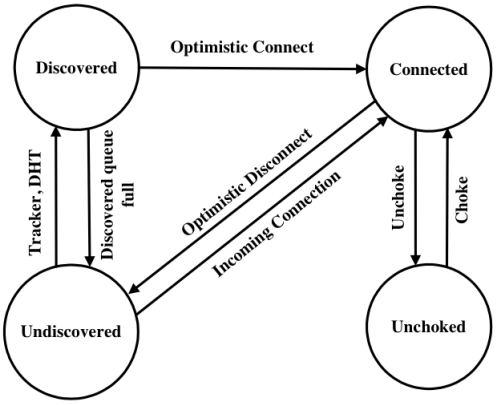

Fig. 2. State Diagram for Connection Management in Azureus. [7]

For understanding what happens we now have to take a look at the connection algorithms.
Azureus accepts received connection requests immediately unless a specific limit is reached (about 50). As shown in figure 2 the new connected peer will change its state from undiscovered to connected. Connected peers occupy a connection slot but do not receive any data until they are unchoked. Furthermore Azureus maintains a queue called 'Discovered Peers' which only contains peers the client discovered by itself using various methods. [7] If the connection list has free slots the oldest peers in the discovered list get included into it. This is called 'Optimistic Connect'. When seeding Azureus keeps track of the last time data was sent to a peer. In order to make room for new peers, every 30 seconds Azureus checks if one of the peers has not received data for more than 5 minutes. If this is the case the peer is disconnected and enters the undiscovered state.
The connection attack on Azureus causes all regular downloads to fail. The reason for this is a combination of the seeding algorithm and the connection algorithm used. Remember: a seeding peer prefers leechers who download at high speed but also have downloaded low amount of data. Because the attacking clients do not download data at all, the second criterion allows them to grab some of the seeders upload slots. As you know the connection algorithm disconnects leechers that have not got data for more than 5 minutes. Because the attackers outnumber the leechers, all the leecher disconnect one after another. Even if attackers get disconnected their chance of getting connected again is high because in contrast to friendly leechers they will steadily try to reconnect.
BitTornado is more robust because it uses a pure bandwidth first algorithm. Therefore the attackers can not get upload slots and fail. Only if the seeds upload bandwidth per slot is very low (so that the download speed of leechers and attackers is limited by the seed) the attack is moderately successful but then it is more like a bandwidth attack and the only advantage the attackers have is their quantity .

B. Attacks On Leechers Or The Whole Swarm

This section will describe attacks that are not specifically targeted on seeders. Still they aim on stopping the distribution of a file completely. This section will start with the sybil attack which can be combined with some other attacks.

1) The Sybil Attack: Usually you would think a peer exists exactly once per swarm he maybe a seeder, a leecher or something else but he only exists once. The Sybil attack [10] [4] tries to take over a swarm by faking multiple identities. In BitTorrent this is possible because identities are generated autonomously by peers, so there usually is no central entity for controlling them although a tracker could do this. Faking an identity is not expensive thus the number of multiple identities could be very high, sybils could even represent the vast majority of a swarm. However just by creating multiple identities BitTorrent is not harmed too much,thus the following attacks described by the authors of [4] try to combine sybils with other techniques. In Chapter 4 of this work sybils will be used for achieving higher download rates.

2) Lying Piece Possession Attack: This attack described in [4] tries to spread false information about piece availability. The goal is to exploit the local rarest first policy used in BitTorrent. Therefore malicious peers announce they have pieces they do not really have. If there are many of those malicious peers all other peers will delay downloading the pieces because they think it is not very rare. The funny thing about this is that in contrast to the peers belief the pieces become increasingly rare and possibly even disappear completely. To make this attack efficient it should be combined with the sybil attack as described above. The effect of this approach is affected by various parameters e.g. the number and real dissemination of pieces lied about, the number of malicious peers and the file size. [4]. The impact evaluation discussed in [4] shows an increasing effectiveness of the attack the higher the number of malicious peers is. Although they note that they could not tell if this is caused by the piece lying or the peer eclipsing effect (as described below). As a matter of fact the lying piece possession attack can be very harmful and may be able causing BitTorrent swarms to fail.
The authors of [11] show a possible countermeasure with low overhead. They therefore introduce an algorithm called 'PeerRotation' which tries to determine malicious peers that seem to be uninterested in exchanging data and replace them with other peers.

3) Eclipse Attack: In A we already discussed the connection attack on seeders, in [4] a similar approach is used to attack a whole swarm using sybils which try to snatch as many connection slots of a peer as possible. As a result of this the friendly peer is eclipsed, he can not connect to other friendly peers anymore because the attackers can mediate most or all communication [4] and so he starves. Figure 3 shows this in a very simplified way. Remember that peers usually have a connection limit of about 50. Attacking peers in contrast can open as many connections as their resources allow them. This means, that with a relatively low number of malicious peers a nearly unlimited number of friendly peers can be eclipsed.

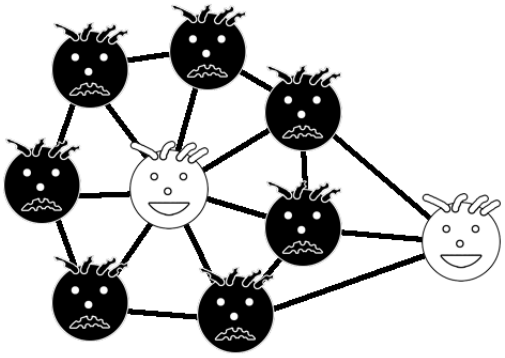

Fig. 3. Eclipse Attack: Friendly peers eclipsed by many malicious sybils

The measured results in [4] show that this attack is highly efficient.

4) Piece attack: The piece attack described in [1] is similar to the Lying Piece Possession attack discussed before. The goal of the piece attack is to slow down the download process. As you know a BitTorrent metafile contains SHA1 hashes for every piece. Those are used for integrity checking. So if a malicious peer would attempt to send fake data the integrity check would fail and the piece would be downloaded again from another peer. The problem in doing so is, that pieces in BitTorrent are split into blocks and blocks from one piece can be received from various peers. As you may remember single blocks can not be integrity checked because hashes are only available for the whole piece (everything else would make the metafiles way too big). So if a malicious peer sends fake blocks the integrity check will not fail until the peer gathered all blocks of that piece. A blacklist of malicious peers could solve this problem, unfortunately detecting who sent the fake block is not possible. P.Dhungel et al. [1] calculate the probability of downloading a clean piece. According to them if n is the number of neighbors that claim to have the piece and m denotes the number of attackers and k denotes the subset of n which is selected by the peer for downloading the blocks, the probability of downloading a clean file is approximately $(1 - \frac{m}{n})^k$. Thus for a successful attack the fraction $\frac{m}{n}$ has to be high. Naturally this value will be higher when peers try to download a rare file. They also show that in the endgame of a torrent the piece attack is most efficient because there only rare pieces are requested.
Countermeasures against this attack are introduced in [1] and [11]. Thereby in [11] a reputation system is suggested which raises a peers reputation if he contributed to correct pieces and lowers his reputation if he contributed to corrupted pieces. If a threshold is undershot the peer will be moved to quarantine. Their results show, that this approach is efficient because it does not introduce any overhead if there is no attack.
The countermeasures suggested in [1] are based on three heuristics. The first heuristic only measures the amount of bad pieces which an IP has contributed to. The second heuristic measures the ratio between good pieces and bad pieces and

the third heuristic measures the ratio of bad pieces a peer participated in and the total number of blocks he contributed. Attackers will try to minimize their bandwidth cost and thus may only send one corrupted block per piece to a peer. Therefore a high ratio in heuristic three could convict them. Measurements in [1] show, that heuristic three is best suited for practical use but it could be combined with heuristic two.

IV. Selfish Behavior In BitTorrent

So far we only discussed attacks with the target to harm the BitTorrent network and prevent files from being downloaded or slow down the downloading process. Now let us shift attention to seemingly contrary topics, namely free riding and selfish peers. As described before BitTorrent uses a tit-for-tat incentive mechanism to ensure fair sharing. Free riding tries to trick that mechanism, so downloading can be achieved without contributing. There are different reasons why one would aspire this. E.g. in some countries downloading copyrighted stuff is not chargeable but uploading is. Slow internet connections with low upload speed could be another reason. [12]
Unfortunately free riding can be very harmful for a P2P system because like the attacks in the last chapter it slows down the whole network. Also users are selfish or just rational, so they want to download at highest possible speed. This chapter is going to show you some attempts to achieve free riding or cheat to achieve higher download rates than usual. Different types of selfish behavior or free riding are described in [2] [12] [6] [5] [3] [13]. this work will refer to some of them. Peers that try to achieve free riding will be called selfish peers.

A. The Large View Exploit

Remember, that if a BitTorrent client connects to a tracker and requests peers the tracker delivers a random subset of peers known to him. Also a peer only holds connections to a maximum of about 50 other peers. Further recall the optimistic unchoking BitTorrent implements and the round robin uploading seeders do.
The large view exploit as described in [12] and used in the BitThief implementation of Locher et al. [13] mainly tries to exploit these points. Therefore the authors of [13] make modifications to an existing BitTorrent client. E.g. their client never uploads, it mimics the behavior of a new client and it requests new peer lists from the tracker every 15 seconds. Also it has an unlimited number of connection slots and thus connects to as many peers as possible. Figure 4 demonstrates the amount of peers known to the tracker and the peers known to a regular client. The large view exploit tries to gather as many of the tracker known peers as possible. The goal is to get often optimistically unchoked and so download at fast rates. If the client is connected to more seeders he also has a higher chance in benefiting from their round robin unchoking. The results of [13] show that with this approach even higher download rates than with a honest client can be reached if the swarm is large enough.

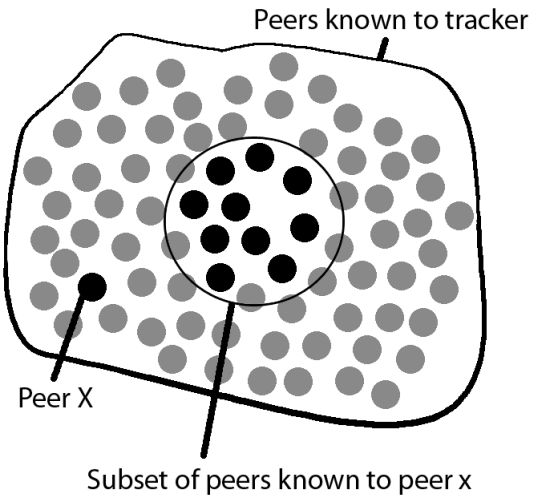

Fig. 4. Peers known to a specific peer vs peers known to tracker

B. Other Optimizations For Achieving Higher Speed

This section will describe a few possibilities to achieve higher download speeds than with regular clients. Some of those approaches could even be combined with the large view exploit to achieve better free riding.

1) Do not download rarest first: The developers of BitThief, a free riding BitTorrent client described in [13] implemented piece selection algorithm that does not apply the rarest first policy but fetches whatever pieces he can get. So it never leaves an unchoke period unused. [13]. This could help to download slighty faster.

2) Only Interact With The Fastest Peers: The authors of [3] describe three exploits for achieving higher download speeds, one of them is to identify the fastest peers and only communicate with them. This also includes to do no optimistic unchoking thus slow peers can not interact with us in any way. The BitTorrent protocol specifies that, every peer should send an advertisement when he finished downloading a piece. Through observing those advertisements it is possible to approximately find the fastest downloading peers of a swarm. In most cases high download speed comes with relatively high upload speed, so those peers are selected. Seeders do not finish any pieces so they do not send advertisements, and thus every pieces from seeds are requested not regarding their speed.

3) Use Sybils: Another good-sounding approach is the use of multiple identities aka sybils. If a single client is connected to a tracker with multiple identities and the tracker delivers those sybils to other peers. The chance of being unchoked could raise. Unfortunately the examinations of [14] show, that this does not result in a higher download speed.

4) Upload Garbage: Because of the tit-for-tat incentives system uploading is rewarded by other peers. Unfortunately we can only upload pieces we already got thus we can not upload to all other peers because they may have those pieces already. To exploit this we could attempt advertising

rare pieces we do not really have (and if requested upload random garbage). This is very similar to the piece attack described in chapter 3. Because the integrity check fails not until all blocks of a piece are gathered by the downloading peer, we could try to upload only some blocks per piece. Additionally in [3] is suggested to not advertise all pieces because then other peers would think we are a seeder and thus not letting us download from them. In theory the peers should not recognize us as malicious peer and so reward us for uploading to them. As the evaluation in [3] and [13] shows this really works for the official implementation. Unfortunately the official implementation is not used much and clients like Azureus are not as easy to fool. E.g. Azureus uses a pretty nice mechanism for preventing this attacks: if an integrity check fails, it looks who contributed with the most blocks to it. Then Azureus requests the missing blocks from that peer. If the peer refuses to answer or sends garbage his IP gets banned.

V. Comparison And Future Work

This work was based entirely on BitTorrent and some of the described problems are very specific. You might be surprised but still BitTorrent is one of the most resilient P2P overlay networks today.

Former networks like FastTrack (used by KaZaA) which was very popular have suffered massive attacks or law suits and thus became nearly unusable or totally disappeared. All of the P2P file sharing networks left today are prone to many different kinds of attacks and selfish peer behavior.

Future protocols have to learn from the mistakes of previous ones. Today there is much scientific research done on the development of robust, yet scalable P2P networks and many of the discussed attacks could be prevented.

During the last years BitTorrent got quite some additions to the original protocol, e.g. distributed hash tables (DHT) were introduced for trackerless communication.

Self-organizing, scalable and robust DHT may play a big role in P2P systems of the future because through their use P2P networks can be freed of trackers or other central servers which are always prone to attacks or law suits. Although DHTs bring many new problems, e.g. how can sybil attacks be prevented if there is no central entity that can assign identities? In papers like [15] fundamental design principle for such DHT based networks are discussed, they will help in further development.

VI. Conclusion

The contribution of this work mainly was to summarize and describe different types of attacks and exploitations that could harm BitTorrent and other P2P file-sharing overlay networks. Some of them are already used massively by attackers. It has been found that BitTorrent is vulnerable and exploitable in many different ways. Still today BitTorrent is one of the most robust public used P2P networks (if not the most robust) and has the potential power to stay alive for a long time. Therefore it is important to reveal and understand possible attacks. Protocol additions like DHT and different implementations of the BitTorrent protocol already exist. To become more robust the official protocol should include workarounds for known attacks, some possible countermeasures were broached in the course of this work. The designers of new P2P overlay protocols (at least those intended for file sharing) should consider all shown problems and learn from them.

Literatur

[1] P. Dhungel, D. Wu, and K. W. Ross, "Measurement and mitigation of bittorrent leecher attacks," *Computer Communications*, July 2009. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2009.07.006

[2] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent," in *In NSDI'07*, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.112.6384

[3] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting bittorrent for fun (but not profit)," 2006.

[4] M. A. Konrath, M. P. Barcellos, and R. B. Mansilha, "Attacking a swarm with a band of liars: evaluating the impact of attacks on bittorrent," *Peer-to-Peer Computing, IEEE International Conference on*, vol. 0, pp. 37–44, 2007.

[5] D. Hales and S. Patarin, "How to cheat bittorrent and why nobody does," in *European Conference on Complex Systems*, 2006.

[6] B. Cohen, "Incentives build robustness in bittorrent," 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.1911

[7] P. Dhungel, X. Heiz, D. Wu, and K. W. Ross, "The seed attack: Can bittorrent be nipped in the bud?" Tech. Rep., 2009.

[8] S. Rouibia, J. Vayn, O. Beauvais, and G. Urvoy-Keller, "Early stage denial of service attacks in bittorrent: An experimental study," in *WETICE '08: Proceedings of the 2008 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 141–142.

[9] Z. CHEN, C. LIN, Y. CHEN, V. NIVARGI, and P. CAO, "An Analytical and Experimental Study of Super-Seeding in BitTorrent-Like P2P Networks," *IEICE Trans Commun*, vol. E91-B, no. 12, pp. 3842–3850, 2008. [Online]. Available: http://ietcom.oxfordjournals.org/cgi/content/abstract/E91-B/12/3842

[10] F. Pontes, F. Brasileiro, and N. Andrade, "Bittorrent needs psychiatric guarantees: Quantifying how vulnerable bittorrent swarms are to sybil attacks," *Dependable Computing, Latin-American Symposium on*, vol. 0, pp. 65–72, 2009.

[11] M. P. Barcellos, D. Bauermann, H. Sant'anna, M. Lehmann, and R. Mansilha, "Protecting bittorrent: design and evaluation of effective countermeasures against dos attacks," in *27th International Symposium on Reliable Distributed Systems (IEEE SRDS 2008)*, October 2008.

[12] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, "Free-riding in bittorrent networks with the large view exploit," in *IPTPS*, 2007.

[13] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *In HotNets*, 2006.

[14] J. Sun, A. Banerjee, and M. Faloutsos, "Multiple identities in bittorrent networks," 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.9160

[15] B. Awerbuch and C. Scheideler, "Towards a scalable and robust dht," 2006.