Attacking a Swarm with a Band of Liars: evaluating the impact of attacks on BitTorrent

Marlom A. Konrath, Marinho P. Barcellos, Rodrigo B. Mansilha

PIPCA – Postgraduate Program on Applied Computing
UNISINOS University – Universidade do Vale do Rio dos Sinos
São Leopoldo, RS – Brazil
marlomk@unisinos.br, marinho@acm.org, rmansilha@gmail.com

Abstract

BitTorrent has become one of the most popular Internet applications, given the number of users and the fraction of the Internet traffic it consumes. Its wide adoption has exposed some potential problems, like selfish peer behavior. Related research efforts so far have focused on modeling the dynamics of swarms, as well as devising incentive mechanisms that improve fairness without sacrificing efficiency. To the best of our knowledge, this is the first paper to evaluate the impact of attacks that exploit BitTorrent vulnerabilities with the sole intention of harming *a swarm. The paper sheds light on BitTorrent behavior by presenting state diagrams, describes two attacks, and then evaluates their negative impact in realistic swarm settings. To evaluate the impact of attacks, a discrete-event simulator was developed and validated against an experimental evaluation performed in a controlled environment. Our findings show the seriousness of the problem and should be the basis for the development of new mechanisms to increase BitTorrent security.*

1 Introduction

Peer-to-Peer (P2P) file sharing has become one of the most relevant network applications, allowing the fast dissemination of content in the Internet. In this category, BitTorrent [3] is one of the most popular protocols. It is used everyday by millions of people to share legal as well as copyrighted content with millions of others [11]. It already consumes a substantial fraction of the Internet traffic [8]. BitTorrent is now being used as the core technology of content delivery schemes with proper rights management that are being put in operation (e.g., Azureus Vuze). Major companies like Warner Brothers, 20^{th} Century Fox and BBC are distributing content through it.

However, as the popularity of BitTorrent grows, so does the risk and impact of malicious attacks exploiting its potential vulnerabilities. This paper identifies attacks to BitTorrent and evaluates their impact on downloading efficiency (and eventual success) of peers taking part on a session (called *swarm*). Understanding vulnerabilities and their negative impact, researchers may propose counter-measures or changes in the architecture in order to improve it. Previous papers focused on understanding BitTorrent from the performance point of view, devising analytical models of BitTorrent and improving its incentive mechanism. The main contribution of this paper is to identify and measure the impact of two attacks to BitTorrent. In order to identify vulnerabilities, we shed light on BitTorrent protocol behavior, contrasting description and traces generated by actual implementations.

The rest of the paper is organized as follows. Section 2 discusses related work, while Section 3 addresses the components and behavior of BitTorrent. In Section 4, we identify attacks performed by malicious users with the intention of either slowing down or completely preventing downloads from happening. Section 5 describes our simulation model, which was implemented and used to evaluate the impact of attacks, as presented in Section 6. In Section 7 we conclude the paper and discuss future work.

2 Related Work

BitTorrent is one of the most popular P2P systems and the literature about it is rich. A related body of work regards *selfish* peer behavior in BitTorrent. Several papers in the literature demonstrate that the incentive mechanism employed by BitTorrent cannot guarantee fairness in sharing, and then propose new interaction mechanisms [18, 9, 10, 15].

Nielson [2] et al. define a taxonomy for rational attacks, in which an attacker wishes to maximize its utility, but does not harm the system unless it increases the benefits. Locher et al. [4] present BitThief, an agent which downloads but never contributes to the network. Several strategies are discussed to improve the performance of a free-rider. Sirivianos et al. [7] identify an attack in which a free-rider profits from having a wider view (in number of peers) of the swarm. The attack is shown to be reasonably successful by means of experiments in PlanetLab. In [20] Piatek et al. present BitTyrant, a modified version of Azureus that adheres to the protocol but uses different policies to improve download performance while reducing upload contributions.

The papers that are closest to ours are [1] and [17]. The former presents two strategies, namely multiple identities and garbage upload, that can be combined by a peer to perform free-riding. The latter presents three techniques in which a peer does not respect the protocol and lies about piece availability or refuses to cooperate.

The common aspect among the previous papers is the focus on how a peer can maximize the benefit received from a BitTorrent swarm whilst minimizing the resources it contributes. In contrast, in our work downloading is not relevant to a peer, whose sole intention is to *hinder* a swarm (using the minimum amount of resources needed). The idea is to make use of false piece announcements and/or large number of sybils to slow down or, ideally, to prevent content from being distributed. In Section 4 we detail two types of attacks (Piece Lying and Eclipse) to BitTorrent, and in Section 6 we evaluate the potential impact they can have on a swarm. Before, we quickly review BitTorrent and present diagrams illustrating its behavior.

3 The BitTorrent Architecture

We now summarize the main elements of the BitTorrent architecture. The *content* (a set of files) to be distributed is described through a meta-data file whose extension is typically .torrent. Such file is created by the user that publishes the content, and contains information like name and size of each file, hashes for data, and one or more IP addresses of *trackers*. A tracker is a central element that coordinates a *swarm* and helps peers to find other peers in the same swarm. Peers can act either as *seeders* or *leechers*, depending on if they have or have not already completed their downloads, respectively. The content published is organized in *pieces*, and these are subdivided into *blocks*. Blocks are the smallest unit of data exchange between a pair of peers. Peers establish a connection with each other and exchange *bitfields* containing information about piece availability. A peer requests to one or more peers the download of blocks that belong to one given piece in which it is interested. Requests are serviced according to a reciprocity-based incentive mechanism, as explained next.

The incentive mechanism is used to increase the cooperation among peers. It works in rounds, typically 10 seconds long. A peer evaluates in a round the three peers that have most contributed to itself and then favors such peers in the next round, serving their requests. The remote peers to whom the local peer is uploading are marked as *unchoked*, while others are set as *choked*. A peer is kept informed by a remote peer regarding how that peer regards this one (changes in state from choked to unchoked and vice-versa). Three peers are marked as unchoked, and a fourth peer is chosen randomly for uploading between the connected ones. The choice of this peer, called *optimistic unchoking*, allows a peer to probe for peers with better download and upload rates.

In the absence of a formal description of the protocol, we analyzed the protocol information given in [6] and traces of popular user agent implementations. The results were state and time diagrams that attempt to define and illustrate the behavior of peers in BitTorrent swarms. Note that even though the diagrams are reasonably accurate, the lack of a formal specification leads to inevitable ambiguity. Figure 1 represents the state diagram, both globally and with emphasis on piece exchange. The main elements are discussed next.

As shown in Figure 1, a peer begins in state START and typically follows the transitions WAIT_TRKCONNECT $\rightarrow$ WAIT_PEERLIST $\rightarrow$ MANAGING_PEERLIST. This means a peer establishes a connection with a tracker (picked from the .torrent file), obtains a list of peers, and then attempts to open connections with a subset of the peers in the list, as well as receives connection requests from remote peers. The diagram follows a concurrent design: for each successfully established connection, a new thread is created (represented with dashed lines) and assigned with the task of handling the communication with the corresponding peer. A thread may begin in state WAIT_PEERCONNECT, if the local peer requested the connection, or in REC_AWAIT_BITFIELD, if the request was remotely originated. In the former case, the state transitions would be WAIT_PEERCONNECT $\rightarrow$ WAIT_HANDSHAKE $\rightarrow$ CONNECTED, whereas in the latter would be REC_AWAIT_BITFIELD $\rightarrow$ CONNECTED.

The state CONNECTED, marked in the diagram, refers to the state variations that a thread may suffer reflecting the interactions with the remote peer that was assigned to the thread. This relation is driven by two state diagrams: *download* and *upload*. The sole connection between the two is the process of choosing peers, according to the incentive mechanism, to whom uploading must be conceded. Figures 2 and 3 show the corresponding diagrams.

Regarding downloads (Figure 2), a thread starts in the state NOT_INTERESTED. Examining the bitmap received from the remote peer, the thread may find that the remote peer has one or more pieces that the local peer does not have. If so, it changes to INTERESTED_CHOKED and informs the remote peer through an **Interested** message; however, the thread is being choked and may not yet request pieces. It will leave the state to INTERESTED-UNCHOKED when (and if) a message **Unchoked** arrives, and thereafter send requests for a block and receive the desired data, until the remote peer starts choking the peer, or the peer completes the piece, in which case it tells other peers about it with a **Have** message.

Regarding uploads (Figure 3), a thread starts in the state NON_INTERESTING, reflecting the fact that this peer is not interesting to the remote peer. When the peer receives a message **Interested**, it changes to the state INTERESTING_CHOKING: the local peer has pieces that are needed by the remote, but is choking the remote peer. The remote peer may send a message to inform the local peer that the remote one is not interested anymore (for example, because the latter obtained all pieces possessed by the local peer). When the

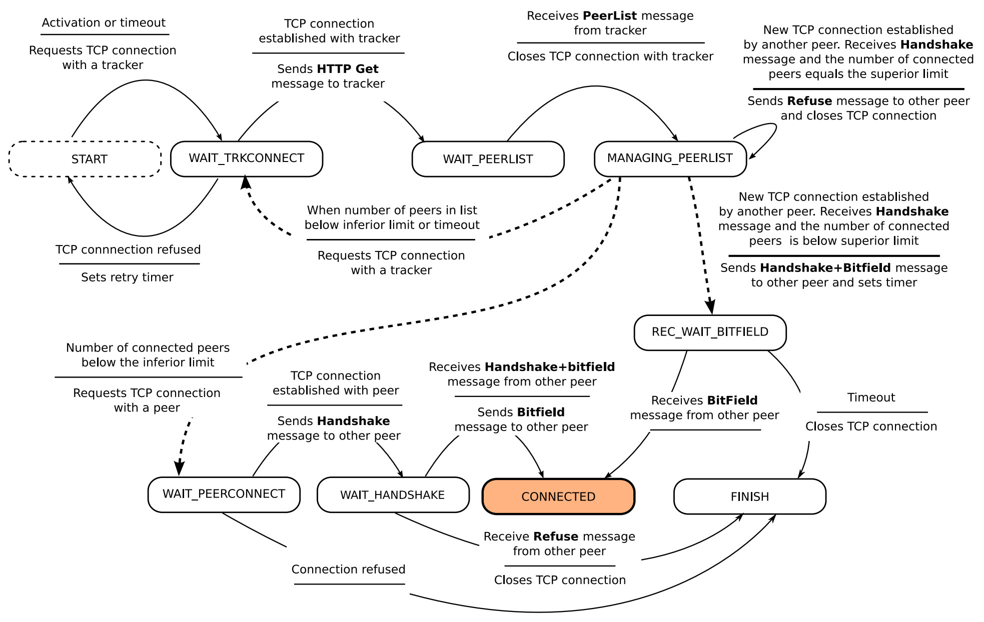

Activation or timeout
Requests TCP connection with a tracker
TCP connection established with tracker
Sends **HTTP Get** message to tracker
Receives **PeerList** message from tracker
Closes TCP connection with tracker
New TCP connection established by another peer. Receives **Handshake** message and the number of connected peers equals the superior limit
Sends **Refuse** message to other peer and closes TCP connection
START
WAIT_TRKCONNECT
WAIT_PEERLIST
MANAGING_PEERLIST
When number of peers in list below inferior limit or timeout
Requests TCP connection with a tracker
New TCP connection established by another peer. Receives **Handshake** message and the number of connected peers is below superior limit
Sends **Handshake+Bitfield** message to other peer and sets timer
TCP connnection refused
Sets retry timer
REC_WAIT_BITFIELD
Number of connected peers below the inferior limit
Requests TCP connection with a peer
TCP connection established with peer
Sends **Handshake** message to other peer
Receives **Handshake+bitfield** message from other peer
Sends **Bitfield** message to other peer
Receives **BitField** message from other peer
Timeout
Closes TCP connection
WAIT_PEERCONNECT
WAIT_HANDSHAKE
CONNECTED
FINISH
Receive **Refuse** message from other peer
Closes TCP connection
Connection refused

Figure 1. A State Diagram for BitTorrent

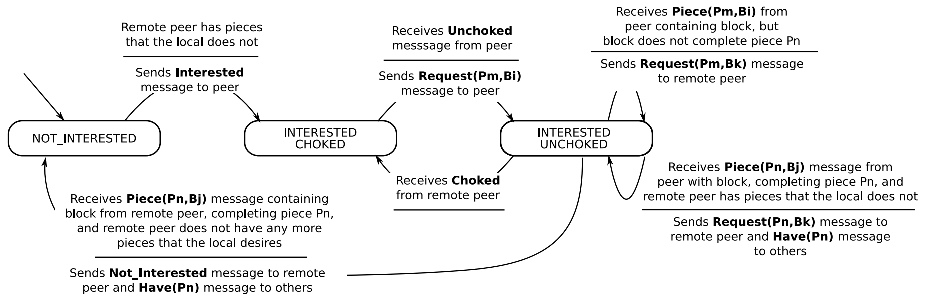

Remote peer has pieces that the local does not
Sends **Interested** message to peer
Receives **Unchoked** messsage from peer
Sends **Request(Pm,Bi)** message to peer
Receives **Piece(Pm,Bi)** from peer containing block, but block does not complete piece Pn
Sends **Request(Pm,Bk)** message to remote peer
NOT_INTERESTED
INTERESTED CHOKED
INTERESTED UNCHOKED
Receives **Choked** from remote peer
Receives **Piece(Pn,Bj)** message from peer with block, completing piece Pn, and remote peer has pieces that the local does not
Sends **Request(Pn,Bk)** message to remote peer and **Have(Pn)** message to others
Receives **Piece(Pn,Bj)** message containing block from remote peer, completing piece Pn, and remote peer does not have any more pieces that the local desires
Sends **Not_Interested** message to remote peer and **Have(Pn)** message to others

Figure 2. State diagrams for threads regarding downloads from remote peers

remote peer is selected for upload, the thread sends **Unchoked** to the remote peer and changes to UNCHOKING. The peer then may receive multiple **Request(Pm,Bj)** messages, replying with **Piece(Pm,Bj)**. If the remote peer is unselected for upload, the thread sends a **Choked** message and returns to INTERESTING_CHOKING.

3.1 Piece Selection Policy

The success of a download (and the whole swarm) is influenced by the piece selection policy employed by the peer. The selection in BitTorrent follows the *Local Rarest First* (LRF) policy, as hinted in Section 2. It is based on the concept of choosing first the pieces that are *less* replicated among peers with whom the peer is connected to; when there is more than one piece with the smallest degree of replication, then the choice is random.

There are three exceptions to this rule: when the peer is beginning its download (situation in which it has less than 4 pieces), successfully finishing it (already requested all the missing blocks) or acting as a seeder (has all pieces).

3.2 Peer Connection Policy

Peers obtain from a tracker the IP addresses from other peers in the same swarm. Peers that take part in the swarm periodically connect to the tracker and this way notify their presence. After a peer connects to the tracker, it informs the number of peers (**numwant**) that it wishes to receive, 50 by default.

In addition to the list of peers, the tracker provides other information including **interval**, the time (in seconds) that the peer must wait before doing another request. This value is also used to determine the time the tracker will keep the IP address of the peer in the list of active ones. If the time exceeds this limit plus a safe margin, the tracker assumes the peer has silently abandoned the swarm. The list of peers kept by a tracker defines who belongs to the swarm, and

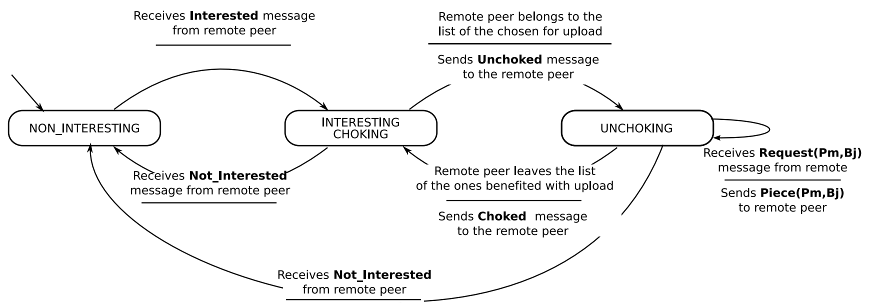

Receives **Interested** message from remote peer
Remote peer belongs to the list of the chosen for upload
Sends **Unchoked** message to the remote peer
NON_INTERESTING
INTERESTING CHOKING
UNCHOKING
Receives **Request(Pm,Bj)** message from remote
Sends **Piece(Pm,Bj)** to remote peer
Receives **Not_Interested** message from remote peer
Remote peer leaves the list of the ones benefited with upload
Sends **Choked** message to the remote peer
Receives **Not_Interested** from remote peer

Figure 3. State diagrams for threads regarding uploads from remote peers

its size can vary from two or three peers to the hundreds of thousands. Upon request, the tracker randomly mounts a list of up to **numwant** peers (which defaults to 50) and sends it to the requesting peer.

There are several situations in which a peer may obtain the identity of more peers. First, when a peer joins the swarm, it sends a message **Get** to the tracker, which responds with **PeerList**. Second, when the list shrinks to less than 20 peers (inferior limit), the peer sends a new **Get** to the tracker. The third situation arises when the peer is contacted by a previously unknown peer in the swarm – i.e. a remote peer that is not currently in the list kept by the local peer. There is also an extension to the protocol that allows a peer to obtain peer IPs from other peers, without involving the tracker. The maximum number of connections that a peer may hold is normally limited to 55. Typically, a peer tries and opens 30 connections to other peers, and the remaining are left for incoming connection requests from other peers.

4 Subversion and Attack Strategies

This section addresses attacks to BitTorrent swarms. First, we discuss the *Sybil* attack in BitTorrent (Subsection 4.1), and then how it can be employed to launch two attacks: piece lying (Subsection 4.2) and the eclipsing of correct peers (Subsection 4.3).

4.1 Sybil

The Sybil attack [12] in P2P networks consists in a single peer presenting itself with multiple, virtual identities, usually with the aim of exploiting reputation-based systems. Using this attack, a malicious peer (henceforth called a *liar*) may get to represent a substantial fraction of the P2P network, and thus compromise it. Douceur suggests the use of certification authorities as the best form of protection against this kind of attack.

In BitTorrent, identities are randomly generated. An attacker, therefore, may exploit this vulnerability and obtain multiple identities. Peers that refuse to cooperate may be banned, but it may not be easy to distinguish peers not intentionally cooperating from peers facing connectivity or overloading problems.

4.2 Lying Piece Possession

As discussed in Section 3, peers employ messages **Bitfield** and **Have** to inform peers about piece possession. Following the LRF policy, (correct) peers strive to increase uniformity in the amount of copies of each piece. A *Piece Lying attack* aims at destroying this balance: a malicious peer does not adhere to the protocol and announces a piece it does not have (thus it is a liar). It artificially increases the level of replication of a piece and therefore induces correct peers to download other pieces first. In other words, the attack consists in turning pieces gradually rare up to the point they disappear from the system, causing swarm failure. As a malicious peer does not wish to deliver the announced piece, it keeps other peers permanently choked.

The attack effectiveness will be affected by the strength of the attack and the state of the swarm when it begins. Relevant aspects include the number of pieces being lied by malicious peers (and the size of the content being shared), the number of malicious peers, the number of correct leechers and seeders, the degree of dissemination of pieces, and if malicious peers are acting in *collusion*.

Also, intuitively it is more effective to lie about a set of pieces than a single one. However, there exists a relation between the size of the set and the effectiveness of the attack: to claim the possession of *all* pieces would not help making one very rare or nonexistent.

The impact of attacks is generally more effective when performed by many peers acting in collusion. In the specific case of making a piece ever rarer, we expect more peers to make the attack more harmful. The more peers lie about a given piece, the more frequent it will *appear* to become, and thus in practice the rarer in fact it will be. These issues are discussed in Section 6.

4.3 Eclipsing Correct Peers

If an attacker has enough physical resources or creates a great number of Sybils, it can attack a swarm using a large number of malicious peers. The attack is made easier by the fact that the same set of malicious peers can be used to potentially attack hundreds or thousands of swarms, provided that only control messages are exchanged between peers.

In an Eclipse attack [19], a set of malicious, colluding peers arranges for a correct node to peer only with members of the coalition. If successful, the attacker can mediate most or all communication to and from the victim. In the context of BitTorrent, the Eclipse attack consists in using a sufficiently high number of malicious peers so that the correct ones connect mostly (or only) with malicious peers. The maximum number of connections a correct peer maintains, by default, is 55. A malicious peer, in contrast, can connect to a larger number of correct peers. Therefore, 55 malicious peers might be sufficient to eclipse any number of correct peers, as long as correct peers only connect to malicious ones.

In practice, however, during an attack the list of peers a correct peer will receive from the tracker (if not itself compromised) will contain both malicious and correct peers. The list is generated randomly every time the tracker is consulted. So, the larger the proportion of malicious to correct peers in the tracker global list, the higher the probability that the list received will only contain malicious peers.

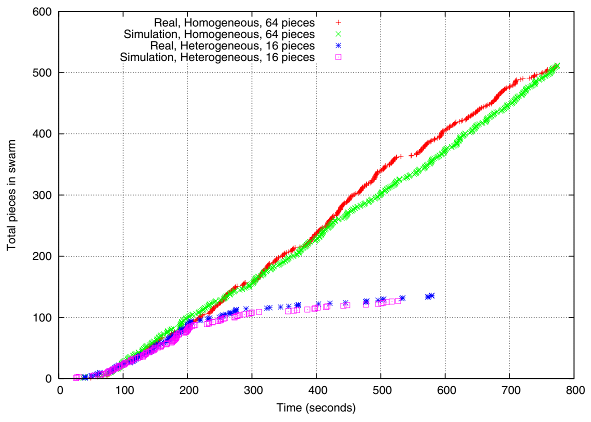


Figure 4. Comparison between real and simulation.

5 BitTorrent Simulation

To evaluate propositions made in the previous section, we devised a simulation model of the BitTorrent protocol. It is based on a combination of the protocol description in [6] and the analysis of packet traces captured from two popular implementations (the overall result is formalized through the state diagrams shown in Section 3). The simulation model was implemented by extending Simmcast, a packet-level simulation framework [5].

Although the model is realistic, several simplifying assumptions were made. First, there is a single tracker, while in actual swarms there may be multiple trackers and peers alternate among them. Second, there is no real data exchange between peers, data messages carrying only control information. A message **BLOCK(p,b)** represents the transfer of a block **b** of a given piece **p**. Third, the simulation assumes TCP as the underlying transport protocol, but does not include details of segment exchange. In the actual implementation of BitTorrent, a block is reliably transferred through a TCP connection to another peer, potentially employing multiple segments, requiring acknowledgments and retransmissions. In the simulation, the **BLOCK** message is sent at once and through a single segment. Application-level flow control is present, though, as a peer is allowed to keep a single outstanding block to a given other peer.

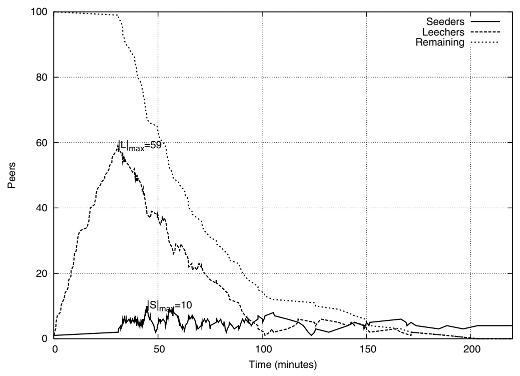
Swarms are made of seeders, leechers and a tracker. Peers can start as seeders (having all pieces) or leechers with no piece. A swarm is created with at least one initial seeder and zero or more initial leechers; all peers that subsequently join the swarm start as leechers without pieces. Leechers will gradually become seeders and eventually leave the swarm. Each leecher has its own arrival time, drawn from a random distribution (such as Uniform, Gaussian or Exponential). Leeching time (also called downloading time) will vary according to multiple factors, including the number of seeders and leechers in the swarm, piece distribution, content size and link capacity. Seeders have a configurable seeding time, which can be individually set either by time or contribution *ratio* (in the latter case a seeder will abandon the swarm as soon as the target ratio is reached). A peer is configured also with the period to the reconnect to the tracker. Other parameters are the number of pieces in the content being shared, the number of blocks per piece, and size of each block. Regarding the network, all peers are interconnected through a cloud. The link capacity between a peer and the cloud can be individually set, both downstream and upstream, in terms of bandwidth and latency.

To validate our simulation model and implementation, we run a set of experiments with two popular BitTorrent agents in a small, controlled environment. We evaluated swarms with different content sizes, piece sizes, bandwidth rates, and number of peers. Packet traces were collected using *passive* monitoring. We then compared the behavior and performance of the simulated and the real runs for configurations of up to 8 peers with both homogeneous and heterogeneous link capacities. To illustrate, consider the similarity in Figure 4; it shows the total number of pieces as the swarm evolves, for two scenarios: homogeneous with 64 pieces and heterogeneous with 16 pieces.

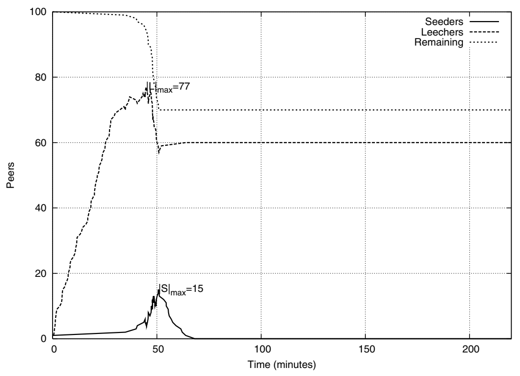
6 Impact Evaluation

In this section we attempt to answer three questions: first, what is the impact of a Piece Lying attack on the dynamics of a swarm, compared when there is no attack; second, what happens when the number of liars increases; third, in the Eclipse attack, which number of malicious peers is more effective and why (what happens during the swarm). Experiments were run multiple times using different seeds for the sake of statistical correctness.

The simulation follows the model described in Section 5. We employed either values mentioned in the literature, like [16], or typically found in real settings. We focus our analysis on the initial hour or two of a swarm. A swarm starts with a single peer, the initial seeder; over time, 100



(a) Normal swarm without attack



(b) Swarm under Piece Lying attack

Figure 5. Normal swarm dynamics and under attack with piece lying

leechers arrive according to a Poisson process, with Exponential inter-arrival times between peers [13, 14]. The attack begins when the 5th leecher arrives at the swarm; malicious peers act in collusion and therefore enter the swarm together. Since in our experiments we focus on the first hours of the swarm (and a hundred peers), we chose the time after the arrival of the 5^{th} peer as compromise between hitting too early (1^{st} peer) and too late (30^{th} peer). The peer-tracker reconnection period is set to 30s. Each time, the tracker sends a list of up to 50 peers.

The experiments are based on a *fair* network, whereby all peers contribute at least the same amount of resources that they have taken (target contribution ratio for initial seeder is 2.0, and for all others, 1.0). Anecdotal evidence suggests that in real-life swarms users are not so altruistic; hence, depending on the kind of content, seeders may leave earlier and under such circumstances attacks would be more successful.

The shared content is of size 640MB, made up by 640 pieces of 64 blocks each, each of which containing 16KB of data. A peer is set with a maximum leeching time, after which it gives up and leaves the swarm (*timed out downloads*); the value used equals is set to be at least ten times the average download in the swarm without attack. A leecher may also abort its download because the entire swarm has failed, situation which arises when any piece reaches zero availability (recall that we assume that peers do not return to the swarm). If the swarm fails, we consider all current and future leechers as *failed downloads*. For the sake of comparison, a lower bound on swarm failures is considered: numeric values for failed downloads are the in complement of 100 leechers.

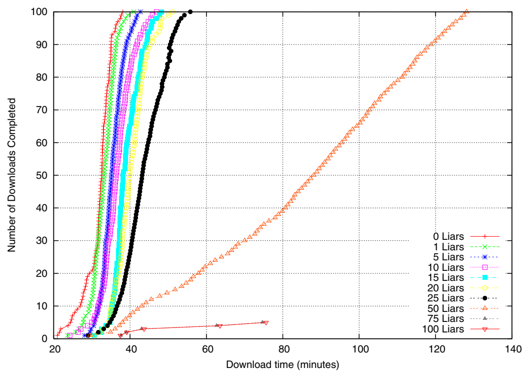
Download and upload bandwidth dedicated to the swarm reflect the asymmetry typically found in broadband networks: 1Mbps and 256Kbps for downstream and upstream, respectively. The set of link pairs is, however, homogeneous: all peers have the same capacity. It is assumed that the tracker is connected through a symmetric, more powerful link, of 4Mbps. Latency between any two peers (and the tracker) is set to 100ms.

As mentioned previously, we expect the effectiveness of an attack to be influenced by the numbers of pieces being lied and peers (denoted as liars or Sybils, depending on the type of attack) involved. To evaluate the effectiveness of an attack, we employed two metrics: the increase of download times, expressed in terms of the cumulative number of completed downloads, and the number of failed or timed out downloads (leecher gives up waiting). When an attack is effective, it will have a large number of timed out or aborted downloads.

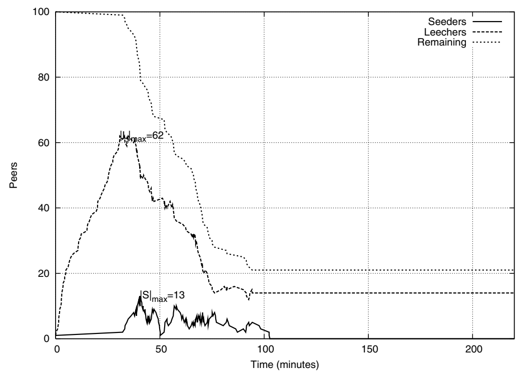
Let $|L|_{max}$ and $|S|_{max}$ denote the maximum number of leechers and seeders recorded, respectively. Considering the scenario previously defined, a swarm will present the dynamics as in Figure 5(a). It represents the arrival of 100 leechers, their gradual transition to seeders, and departure from the swarm in around 200 min. There are three curves: number of leechers, of seeders, and remaining downloads. The latter represents the number of downloads needed to reach the total number of downloads (100), so the experiment finishes when this curve reaches 0 (or, as previously mentioned, when availability for any piece reaches 0). Figure 5(a) shows a *flash crowd* of leechers arriving at the beginning of the swarm, up to when $|S|_{max} = 59$. At that time, leechers start turning seeders, and the Remaining curve drops. This remains so until around 100 min, when the crowd of leechers has diminished and a steady rate of leechers arrive; compensating these new seeders, on their turn, there is a steady rate of leechers leaving the swarm as they contribute until reaching ratio 1.0.

Figure 5(b) shows the same scenario but under attack by a collusion of 25 liars that falsely announce the possession of (the same) 4 pieces. Although the attack starts early in the swarm life, its effects are felt only nearly at 50 min. From this point on, no progress is done – no additional downloads are completed. Notice that $|L|_{max} = 77$, higher than in the case without attack; this is because leechers are tricked and prevented from evolving to seeders due to the attack.

Figure 6(a) illustrates the impact of an attack whereby different numbers of malicious peers lie about 32 pieces, turning them generally rarer. It is clear from the figure that

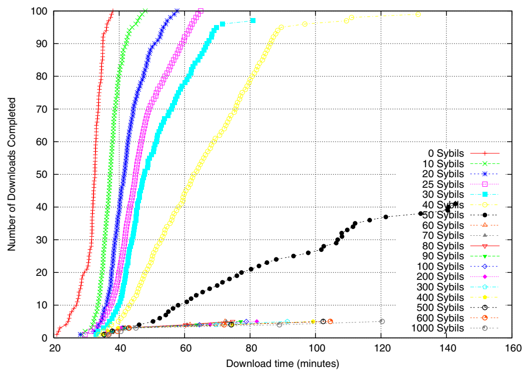


(a) Effect of varying number of liars

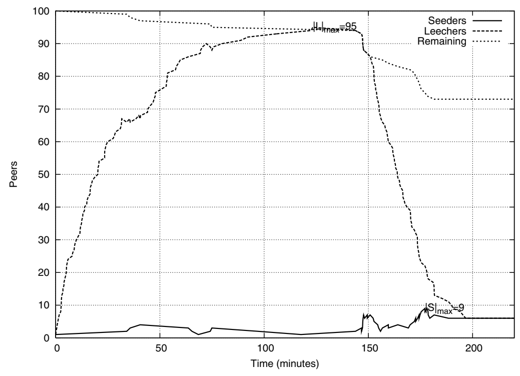


(b) Impact of 10 liars on a given swarm (32 pieces)

Figure 6. Impact of the number of liars



(a) Effect of varying number of eclipsing Sybils



(b) Impact of eclipsing Sybils

Figure 7. Impact of Sybils in Eclipse attacks

for sets of up to and including 25 liars, the delay is negligible. However, some of the downloads failed or timed out, as detailed below. The diagonal curve that corresponds to 50 liars is a clear divisor: from 50 onwards, the attack adds substantial delays (a download can take as much as three times longer). We found that with 75, 100 or more malicious peers the Piece Lying attack was very effective and led to frequent general swarm failure at early stages. Less than 10 peers managed to complete their download successfully. However, given the proportionally high number of malicious peers involved, we could not tell if the effect was more due to the Piece Lying attack or the peer eclipsing effect.

Figure 6(b) shows in detail a particular swarm where 10 peers lie about 32 pieces. Out of 100 peers, 79 completed their downloads, whereas 21 did not. This happens because one or more pieces being lied became unavailable, as correct leechers employed the LRF policy to select pieces considered to be rarer. After some time (around 40 min), the initial crowd of leechers became one of seeders, reached their ratio of 1.0 and then left the swarm. Around 100 min the average number of leechers had dropped enough and there were no more seeders. Consequently, the swarm stagnated.

In the next experiment, the impact of an Eclipse attack is evaluated according to the number of malicious peers. Figure 7(a) illustrates the impact in terms of cumulative downloads, for Sybils set sizes between 10 and 1000. Results indicate that there is a noticeable delay in all cases, which becomes stronger from 40 Sybils onwards. As malicious peers are acting in collusion, so once a correct peer attempts to establish a connection with a malicious one, the latter will inform other peers (Sybils) about that. With 50 Sybils or more, the peer list returned by the tracker tends to be dominated by malicious peers; therefore, the Eclipse attack becomes very effective: less than 50% of the peers complete their download. From 60 liars onwards, the seeder and most peers become eclipsed, and all swarms are subject to failure (all remaining downloads timeout or fail). Consequently, only some of the early correct peers were able to successfully complete their download (around 5).

Figure 7(b) shows the dynamics of a swarm when it is subject to an Eclipse attack of 50 Sybils. The maximum number of leechers reaches a staggering $|L|_{max} = 95$, as the flash leecher crowd almost entirely fails to complete any download and become seeder. The curve intersects with Remaining downloads at around 140 min, when leechers start giving up (i.e., timing out) and abandoning the swarm. The number of leechers drops over 50 min to around 6. Even so, $|S|_{max}$ reaches 9, which means 91 downloads have timed out or failed due to this attack. In other words, this kind of attack is very effective.

7 Concluding Remarks

The contribution of this paper is two-fold. First, different from previous work, it identifies and evaluates the impact of two attacks that strive to cause a BitTorrent swarm to fail. It shows that even modest amounts of resources can be used to attack and hinder BitTorrent swarms, both with Piece Lying and Eclipse attacks. Second, to increase understanding about BitTorrent, state diagrams were created according to the protocol informal description and packet traces. A simulation model was devised and implemented, and its results were validated against real experimental results.

Results indicate that BitTorrent is susceptible to attacks in which malicious peers in collusion lie about the possession of pieces and make them artificially rarer. This attack is effective in delaying or failing downloads when there are around 50 peers or more performing it, for swarms similar to the one considered in our study. Additional results were omitted due to lack of space, but we also evaluated factors like the number of pieces being lied and the time the attack starts. When between 4 and 64 pieces are lied, the attack is more successful (16 being the "best" one for the settings considered); as expected, the earlier the attacks, the more harmful they are. Further, in scenarios where a large number of peers refuse to collaborate, and correct peers may be eclipsed by malicious ones, the effects are much more damaging.

This paper can be extended in four ways. First, the analysis impact can be enhanced by evaluating other scenarios, in particular ones with larger numbers of peers. Second, there are other vulnerabilities in the protocol and implementations. Third, the paper must lead to the design of mechanisms or modifications to BitTorrent to eliminate or reduce the impact of such attacks – this represents a good research challenge. Last, although we have validated the results produced by our simulation against packet traces collected from two popular implementations in a controlled environment, we plan to implement these attacks in a BitTorrent agent so that we can measure more precisely delays induced in real life swarms.

References

[1] *Faithfulness in internet algorithms*, Portland, OR, USA, September 2004. ACM SIGCOMM.

[2] *A Taxonomy of Rational Attacks*, Ithaca, NY, USA, February 2005. Springer Berlin / Heidelberg.

[3] BitTorrent website. http://www.bittorrent.com/, April 2006.

[4] *Free Riding in BitTorrent is Cheap*, Irvine, CA, US, November 2006.

[5] Simmcast: an object-oriented simulation framework for protocol and network research. http://inf.unisinos.br/ Simmcast/, December 2006.

[6] Bittorrent protocol specification v1.0. http://wiki.theory.org/BitTorrentSpecification, May 2007.

[7] *Free-riding in BitTorrent with the Large View Exploit*, Bellevue, WA, US, February 2007.

[8] M. Barbera, A. Lombardo, G. Schembra, and M. Tribastone. A markov model of a freerider in a bittorrent P2P network. In *IEEE Global Telecommunications Conference (GLOBECOM '05)*, volume 2, pages 985–989, St. Louis, MO, USA, November 2005.

[9] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Some observations on bittorrent performance. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS 2005)*, volume 33, pages 398–399, New York, NY, USA, June 2005. ACM Press.

[10] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Understanding and deconstructing BitTorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, March 2005.

[11] S. Das, S. Tewari, and L. Kleinrock. The case for servers in a peer-to-peer world. In *2006 IEEE International Conference on Communications*, volume 1, pages 331–336, Washington, DC, USA, June 2006. IEEE Computer Society.

[12] J. R. Douceur. The sybil attack. In *1st International Workshop on Peer-to-Peer Systems*, pages 251–260, Cambridge, MA, USA, March 2002.

[13] K. Eger and U. Killat. Bandwidth trading in unstructured p2p content distribution networks. In *6th IEEE International Conference on Peer-to-Peer Computing, 2006 (P2P 2006)*, pages 39–48, Washington, DC, USA, September 2006. IEEE Computer Society.

[14] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *Internet Measurement Conference (IMC '05)*, pages 35–48, 2005.

[15] S. Jun and M. Ahamad. Incentives in BitTorrent induce free riding. In *ACM SIGCOMM Workshop on Economics of Peer-to-Peer systems (P2P-ECON)*, pages 116–121, 2005.

[16] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest First and Choke Algorithms Are Enough. Technical report, INRIA, June 2006.

[17] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting bittorrent for fun (but not profit). In *5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, February 2006.

[18] D. Purandare and R. Guha. Preferential and strata based p2p model: Selfishness to altruism and fairness. In *12th International Conference on Parallel and Distributed Systems, 2006. ICPADS 2006*, volume 1, pages 561–570, July 2006.

[19] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *25th Conference on Computer Communications (INFOCOM 2006)*. IEEE, 2006.

[20] USENIX. *Do incentives build robustness in BitTorrent?*, Cambridge, MA, April 2007.