Mistrustful P2P: Privacy-preserving File Sharing Over Untrustworthy Peer-to-Peer Networks

Pedro Moreira da Silva*, Jaime Dias*, Manuel Ricardo*
*INESC TEC, Faculdade de Engenharia, Universidade do Porto
Rua Dr. Roberto Frias, 378, 4200-465 Porto, Portugal
Email: {pmms, jdias, mricardo}@inesctec.pt

***Abstract*—Peer-to-Peer networks are extensively used for large-scale file sharing. As more information flows through these networks, people are becoming increasingly concerned about their privacy. Traditional P2P file sharing systems provide performance and scalability at the cost of requiring peers to publicly advertise what they download. Several P2P privacy-enhancing systems have been proposed but they still require peers to advertise, either fully or partially, what they download. Lacking alternatives, users have adopted anonymity systems for P2P file sharing, misunderstanding the privacy guarantees provided by such systems, in particular when relaying traffic of insecure applications such as BitTorrent.**

Our goal is to prevent any malicious peer(s) from ascertaining users' content interests so that plausible deniability always applies. We propose a novel P2P file sharing model, Mistrustful P2P, that (1) supports file sharing over open and untrustworthy P2P networks, (2) requires no trust between users by avoiding the advertisement of what peers download or miss, and (3) still ensures deterministic protection of user's interests against attacks of size up to a configured privacy protection level. We hope that our model can pave the ground for a new generation of privacy-enhancing systems that take advantage of the new possibilities it introduces. We validate Mistrustful P2P through simulation, and demonstrate its feasibility.

I. Introduction

Peer-to-Peer (P2P) networks are extensively used for large-scale file sharing. As more information flows through these networks, people are becoming increasingly concerned about their privacy. The reasons behind the privacy concerns may be various such as (1) avoiding user profiling, tracking and data mining, (2) engaging in legal content sharing that may be embarrassing or deplorable from a political, religious or social point-of-view, or (3) engaging in illegal or incriminating content sharing.

Traditional P2P file sharing systems are designed for performance and scalability. These systems take advantage of the large number of interconnected peers[1], and their idle resources, to more efficiently distribute contents at the cost of requiring peers to publicly advertise what they download. Given that peers form interest-based communities [6], every single connection presents an opportunity for a malicious peer to passively obtain additional information that may enable user's content interests identification.

Several P2P privacy-enhancing systems have been proposed, such as [17], [19], [12], [13], [10], the majority employing either techniques to provide anonymity, such as *onion routing* [11] and *information slicing* [13], or employing techniques to provide plausible deniability, such as *request relaying* – peers relay requests to create uncertainty about communicating endpoints –, and *content interest disguise* – peers download additional contents to hide their real interests. All these solutions share one common issue: they require peers to advertise, either fully or partially, what they download. Lacking alternatives, users have adopted anonymity systems for P2P file sharing [4], misunderstanding the privacy guarantees provided by such systems [5], in particular when relaying traffic of insecure applications [14], i.e., applications that disclose sensitive information.

Our objective is to prevent any malicious peer(s) from ascertaining the interests of any user downloading a content, either through observation or through active probing attacks, while completing the download in a timely manner. Users interested in downloading contents are provided with plausible deniability against regular peers or groups of colluding peers.

In this paper, we propose a novel P2P file sharing model, which we name Mistrustful P2P, that enables file sharing over open and untrustworthy P2P networks (networks in which peers should be mistrusted) without disclosing user's interests. Our model does not require trust between users by avoiding the advertisement of what peers are downloading or missing. The Mistrustful P2P model ensures deterministic protection of user's interests from regular peers or groups of colluding peers of size up to a privacy protection level configured by the user. It resorts on erasure coding to avoid advertising what is downloaded. The remaining of this paper is structured as follows. Section II details the problem we aim to solve. Section III presents the related work. Section IV provides the required background. Section V depicts the novel P2P file sharing model we propose. Sections VI and VII describe, respectively, the validation of our model through simulation, and the results obtained. Section VIII presents the conclusions.

II. Problem Description

One privacy aspect that is especially sensitive to users is the concealment of their interests. Users look for a privacy-enhancing system that is able to protect their interests from other participants in the system without compromising performance. Also, providing a configurable per content privacy protection level, supporting untrustworthy P2P networks, and

[1]We say *peer* to refer to the network node, and *user* to refer to the person.

ISBN 978-3-901882-84-5, IFIP Networking 2016

having high performance, are all desirable features because user's privacy requirements are idiosyncratic, they may find themselves in need to join untrustworthy P2P networks, and want to download contents as fast as possible. Above all, users tend to prefer security mechanisms that provide strong defense against well defined attacks, even if narrow ones, rather than broad but weak security defense mechanisms.

We consider an attacker that participates in the system, either a regular peer or a group of colluding peers, but not external entities monitoring all traffic of a peer, such as ISPs, or controlling the whole network, such as governments. Protection against link monitoring could be achieved by encrypting communications between peers, but requires key exchange and distribution mechanisms, which are out of the scope of this work.

P2P privacy-enhancing systems typically either completely hide user's activities through anonymity or disguise them by relaying traffic and/or generating cover downloads. Anonymity systems, being Tor [10] the most popular, as a rule, are not designed for P2P file sharing. Nevertheless, lacking alternatives, users have adopted anonymity systems for such end [4], misunderstanding the privacy guarantees they provide [5], and unaware of the privacy impact that relaying traffic of insecure applications, such as BitTorrent, introduces [14]. On the other hand, systems disguising user's activities are designed for P2P file sharing but require users to publicly advertise what they download, either fully or partially, so that peers know to whom blocks (*chunks*, using BitTorrent terminology) can be requested, and also to improve content availability by providing incentives to download rarer blocks. However, advertising what is downloaded (block advertisement) enables download progress tracking, allowing passive attackers to differentiate genuine from cover traffic, therefore disclosing user's interests. Content interest disguise systems that fully download cover contents are an exception to this, but increase greatly the network overhead.

Thus, the problem we aim to solve is how to enable P2P file sharing so that (1) block advertisement and trust links between users are avoided, (2) users are protected against attacks of size for a privacy protection level that is flexible and configurable per content, and (3) contents can be downloaded in due time.

III. Related Work

Several P2P privacy-enhancing systems have been proposed in the literature providing different degrees of privacy to users, the majority of which provides either anonymity or plausible deniability. Tor and Freenet [1] are probably the most prominent anonymity solutions for, respectively, low-latency anonymity and anonymous content distribution networks. Given that anonymity systems tend to introduce more overhead, and do that without improving the overall performance, herein, we depict the state-of-the-art P2P privacy-enhancing systems providing plausible deniability and designed specifically for P2P file sharing.

BitBlender [3] provides plausible deniability by introducing relay peers that simply proxy requests on behalf of other peers. Peers willing to act as relay peers can register at a central node called *blender*, and, once requested, will join a P2P swarm (group of peers sharing a content) in a probabilistic way so that they cannot be distinguished from regular peers. The joining probability of relay peers is defined by the *blender*, when asking registered peers to join a P2P swarm, so that the set of relay peers remains unknown while having the cardinality requested by the *tracker*. As so, BitBlender requires users to trust both the *tracker* and the *blender*.

SwarmScreen [6] provides plausible deniability by obscuring user's interests through cover traffic (content interest disguise). The devised scheme, which consists in "adding a small percentage (between 25% and 50%) of additional random connections that are statistically indistinguishable from natural ones", thwarts guilt-by-association attacks, i.e., attacks in which the user's interests can be inferred with high certainty just by classifying peers based on the behavior of the communities they participate in. SwarmScreen's attack model only considers passive attacks, it is vulnerable to active attacks.

OneSwarm [12] attempts to be an alternative to BitTorrent, and builds upon friend-to-friend networks – networks in which peers only communicate with trusted peers (friends). It provides a high privacy protection level and extensive control over what information is disclosed to other peers. Nevertheless, content availability may be limited as it is difficult to connect any pair of peers using just trusted links. Also, the problem of providing such privacy guarantees in large groups of untrusted peers remains unsolved.

The BitTorrent Anonymity Marketplace [16] follows SwarmScreen's approach to provide plausible deniability. However, in order to protect against both passive and active attacks, all contents are fully downloaded because peers advertise what they download. The authors define *k-anonymity* as the privacy protection level obtained from fully downloading k contents. Thus, as it increases greatly the network overhead, it either prevents downloads from completing in due time or constrains the privacy protection level.

Petrocco et. al [17], following SwarmScreen's approach, proposed a system that aims to protect user's interests without compromising download completion in due time. Their system relies on private swarms, *request relaying*, caching, and partial advertisement of downloaded blocks. As stated by the authors, private swarms are required to ensure a good level of privacy. Yet, to obtain the credentials needed to join a private swarm, peers must trust one or more participants. Also, as only a fraction of the blocks are advertised, it is not clear how a content sharing is bootstrapped with few seeders nor how request relay should operate during periods of content unavailability.

IV. Erasure Codes

Erasure codes are a class of Forward Error Correction (FEC) codes for the Binary Erasure Channel (BEC), a channel in which transmitted data packets are either correct or missing (erasures). Networking layers above the data link layer behave as an erasure channel since packets are either correct, and are delivered, or present errors, and are discarded.

An erasure code generates a set of n symbols from a set of k symbols, $k < n$, at a rate given by k/n, so that any subset of $k(1+\epsilon(k))$ is enough to reconstruct the original information, where $\epsilon(k)$ is the erasure coding overhead. Erasure codes are usually classified according to three orthogonal properties: (1) systematicity, (2) rate fixedness, and (3) coding overhead. An erasure code is systematic if the input symbols are embed into output symbols, and non-systematic otherwise. If n is static and needs to be known before encoding, the erasure code is fixed-rate. If n can be dynamically increased and the amount of symbols that can be generated does not impose any practical limitation, the erasure code is rateless. Finally, an erasure code is said MDS (Maximum Distance Separable) if any k symbols out of n are enough to reconstruct the original information $[\epsilon(k) = 0]$, or non-MDS if additional symbols are required $[\epsilon(k) > 0]$. Non-MDS erasure codes reduce significantly the encoding and decoding time complexity orders by introducing coding overhead.

For P2P file sharing, MDS erasure codes are more suitable as the network is typically the most constrained resource, not the CPU [15]. Non-systematic erasure codes may have the property of only granting access to any part of a content after fully downloading it. Rateless erasure codes enable the setting of n as a function of hard to predict dynamic variables, such as peer arrival rate, to continuously adjust it to the P2P file sharing dynamics.

V. Mistrustful P2P Model

In this section, we describe the Mistrustful P2P model, a novel P2P file sharing model that (1) supports file sharing over open and untrustworthy P2P networks, (2) requires no trust between users by avoiding the advertisement of what peers download or miss, and (3) still ensures deterministic protection of user's interests, through plausible deniability, against attacks of size up to a configured privacy protection level. We consider that the burden of an increased privacy protection level should be on the peer requiring it and not on other peers' resources, thereby peers communicate through direct links, i.e., there is no peer relaying. For this reason, our model relies on cover downloads to protect user's interests, and therefore, targets the development of content interest disguise systems.

The description of each component of the model is conceptual but we provide the instantiation used for validating the Mistrustful P2P model (Section VI) as an example. We hope that our model can pave the ground for a new generation of privacy-enhancing systems that take advantage of the new possibilities it introduces.

A. Overview

Mistrustful P2P avoids block advertisement, and therefore peers no longer know to whom blocks can be requested nor can request a specific block they need. Consider a content divided into k blocks, and that a block request is sent to a randomly selected peer which offers a randomly selected block it owns. Such approach enables to share some blocks between peers but is unfeasible for fully downloading contents because, assuming an uniform distribution of blocks among peers, the probability of obtaining the last block is just $1/k$. Using erasure codes we are able to generate a set of n blocks so that any subset of k' blocks enables to retrieve the content, where $k' = k(1+\epsilon(k))$, and $\epsilon(k)$ is the erasure coding overhead. As so, for the same conditions, the probability of retrieving the last block increases to $1 - \frac{k'-1}{n}$.

Peers only share erasure coded blocks to ensure that access to any part of a content is only granted after fully downloading it, albeit all contents being publicly available. This way, there is no proof that a user had full or partial access to a particular content, including cover ones, by just having downloaded some blocks; thus, they can still participate in its sharing. We assume that this property is provided by the erasure codes, although content encryption can be used to achieve the same goal. Unless otherwise stated, from now on, we say block to refer to erasure coded block.

The Mistrustful P2P model aims at enabling P2P file sharing in large groups of untrusted peers, thereby, no trust links between peers are required. Attending to the idiosyncrasy of user's privacy requirements and to the flexibility required to not constrain the privacy-enhancing systems that can be built on top of our model, the user privacy protection level is configurable per peer and per content. It is defined as a two-dimensional variable composed by c, the size of the largest colluding group considered by the user, and m, the maximum number of blocks that can be shared with any set of c peers, where $c \leq m$ and $m < k$ so that the content cannot be fully downloaded from a single considered colluding group. Thereby, our model provides a deterministic protection of user's interests as long as the effective size of the largest colluding group does not exceed the one configured. When the user privacy requirements for a particular download are not met, the download pauses until they are met again.

Peers, per content, can take one of two roles depending on their privacy requirements and the way they contribute to the file sharing: *seeder* – peer having a content that wants to share, and willing to forgo its privacy –, or *commoner* – peer willing to download content blocks if its privacy requirements can be met. *Seeders* may be the authors or some party interested in publishing the content, and therefore do not require interest's concealment. We consider that there is always at least one *seeder* to ensure content availability, which provides a new erasure coded block for each request it receives. This is a realistic assumption given that a small fraction of publishers are responsible for 67% of the published content and 75% of the downloads in BitTorrent [7]. *Seeders* only refuse to serve block requests if they have no resources available. On the other hand, *commoners* do not create new blocks and only share them if their privacy remains protected. They can either act as an helper (cover downloads) or as, using BitTorrent terminology, a *leecher* (genuine downloads). *Commoners* keep track of what they have shared with other peers both for privacy enforcement reasons and to avoid offering a block twice to the same peer. They may refuse to serve block requests (1) due to resources constraints, (2) if they have

no useful blocks to offer due to privacy constraints, or (3) due to content disguise strategies. The reason behind is never disclosed. If a block is offered, the requesting peer can then either cancel the request (duplicate block) or proceed with its download (useful block).

Let k be number of blocks required to fully download a content, n the number of unique blocks that can be generated, n_a the number of unique blocks available for download, and D_i the set of unique blocks already downloaded by *commoner* i. *Commoner* i attempts to select a random peer that maximizes the probability of having it offering a block that is not in the set D_i. Given that any subset of k' blocks out of n_a enables to fully retrieve the content, increasing n_a maximizes the probability of a peer obtaining a useful block; which can be achieved by increasing the number of *seeders*. However, determining which block should be offered in reply to a request, when should a request be sent, and to which peer is not trivial. Let us consider $e_{i,j}$ to be the number of blocks that can be exchanged between *commoners* i and j, and E_i the number of blocks that *commoner* i can exchange with all other peers (available requests), which is limited by the privacy constraints. If *commoner* i makes too much block requests, more block requests will fail to retrieve useful blocks and it will run out of available block requests ($E_i = 0$); on the other hand, if *commoner* i makes too few block requests, more block requests could have been sent and the available requests to *commoner* j will still be far from zero once *commoner* j leaves ($e_{i,j} \gg 0$).

We devised three mechanisms which main purpose is to attend the issues stated above. The block selection mechanism is used by *commoners* to determine which block should be offered to a requesting peer. The request backoff mechanism aims at delaying block requests to help maximizing the amount of useful blocks that can be obtained from the available block requests, in the shortest time frame possible. The peer selection mechanism aims at determining the peers that should be selected to minimize the download time.

In sum, the Mistrustful P2P model relies on cover downloads to protect user's interests, and has five main components: erasure codes, the privacy enforcement mechanism, the request backoff mechanism, the peer selection mechanism, and the block selection mechanism. It is out of the scope of this work to provide optimal instantiations of each component. We provide only, as an example, the instantiation used for validating our model.

B. Erasure Codes

Although other erasure codes can be used, we refer the reader to [9] for a rateless MDS construction of Reed-Solomon codes that we developed for our model. These erasure codes are defined over the finite field $\mathbb{F}_{p^2}$, where p is a Mersenne prime ($p = 2^q - 1$), and $n \leq 2^{q+1}$. Their performance was evaluated over $\mathbb{F}_{(2^{31}-1)^2}$, so $n \leq 2^{32}$, and does not impose any constraints to the file sharing. Also, they are non-systematic erasure codes that have the property of only granting access to any part of a content after fully downloading it.

C. Privacy Enforcement

The privacy enforcement mechanism ensures deterministic protection of user's content interests, through plausible deniability, against attacks of size up to a configured privacy protection level. Mistrustful P2P guarantees that any peer or colluding group, with size up to c peers, are unable to (1) prove that the user downloaded a particular content or had full or partial access to it, and to (2) distinguish between cover and genuine downloads by tracking its progress. The user can configure, per content, the size of the largest colluding group to consider, c, and the maximum amount of blocks that can be shared with any set of c peers, m, where $c \leq m$ and $m < k$ so that the content cannot be fully downloaded from a single group of size c. The protection provided is guaranteed as long as the effective size of the largest colluding group does not exceed the one considered by the user. Given that finding the maximum intersection between the set of blocks exchanged with any c peers is an NP-hard problem [18], we devised a conservative yet efficient algorithm to evaluate the numbers of blocks that can still be shared with a peer. The algorithm is divided into two main functions, one to update the counter of blocks shared with a peer (Function 1), and the other to determine the number of blocks that can still be exchanged with a peer (Function 2).

Function 1 Update Blocks Shared

```
                ▷ commoners is an array sorted by blocks shared.
        ▷ blksShared is the max no. of blocks shared w/ c peers.
function IncrementBlocksShared(id)
    i ← commoners.getIndex(id)
    if invalidIndex(i) then                                  ▷ New.
        commoners.push(id)
        commoners.last.blks ← 1
        i ← commoners.getIndex(id)
    else                                                     ▷ Known.
        commoners[i].blks ← commoners[i].blks + 1
        j ← i − 1
        while validIndex(j) do
            blksI ← commoners[i].blks
            blksJ ← commoners[j].blks
            if blksI > blksJ then                            ▷ Still unsorted.
                swap(commoners[i], commoners[j])
                i ← j
                j ← j − 1
            else                                             ▷ Sorted.
                break
            end if
        end while
    end if
    if i < c then                                ▷ Changes on top c peers.
        blksShared ← blksShared + 1
    end if
end function
```

Function 1 relies on an array sorted by the number of blocks shared with a peer, and the maximum number of blocks shared with any set of c peers. The function receives a peer id as a parameter, and starts by checking if any blocks have been

exchanged with that peer. If it is the first one, sets the number of blocks shared to 1. Otherwise, the number of shared blocks is incremented, and, if needed, some elements are swapped until the array is again sorted. The maximum number of blocks shared with any set of c peers is incremented if the update was in one of the top c positions of the array. Function 1 has linear time complexity.

Function 2 Blocks to Share Left

```
                ▷ commoners is an array sorted by blocks shared.
        ▷ blksShared is the max no. of blocks shared w/ c peers.
function BLOCKSHARELEFT(id)
    if c > m or m ≥ k then                                  ▷ Invalid.
        return 0
    end if
    top ← min(c, commoners.length)                          ▷ Top peers.
    left ← m − blksShared − (c − top)
    i ← commoners.getIndex(id)
    if invalidIndex(i) then                                 ▷ New.
        if commoners.length > c then
            left ← left + commoners.last.blks − 1
        end if
        return left
    else                                                    ▷ Known.
        if i ≥ c then
            left ← left + commoners.last.blks
            left ← left − commoners[i].blks
        end if
        return left − 1
    end if
end function
```

Function 2 relies on the same variables as Function 1, and also receives the same parameter. It starts by checking if the configured privacy protection level is invalid. If it is valid, $left$ contains the number of blocks that can still be exchanged, ensuring that at least one block is exchanged with each one of the top c peers. This value needs to be updated if there are already at least c peers and the peer referred by id is outside of that set. Function 2 runs in logarithmic time.

D. Block Selection

The block selection mechanism is used by *commoners* to determine which block should be offered to a requesting peer. It plays an important role on how the blocks end up distributed among peers, affecting the probability of peers obtaining useful blocks. This mechanism ensures that no block is offered twice to the same peer, and determines when requests should be refused due to the lack of useful blocks to share.

Although this mechanism should use content sharing information as input, such as the number of requests that end up canceled (both as source and destination), for validating the model we select blocks randomly due to its simplicity. With Mistrustful P2P model there is no need to suddenly terminate, remove downloads, or stop sharing because the privacy protection level does not depend on the time a peer keeps sharing a content, as long as cover and genuine downloads are treated in the same way.

E. Request Backoff

The request backoff mechanism aims at delaying block requests to help maximizing the amount of useful blocks that can be obtained from the available block requests, in the shortest time frame possible. It does so by constraining the set of peers to which block requests can be sent (eligible peers), and by determining for how long no block requests should be performed. Therefore, as the former is a direct result of individual peer behavior and the latter depends on the swarm behavior, we define the backoff time has a two-dimensional variable that has a per peer and a swarm components. The peer backoff component provides the delay to return a peer to the set of eligible peers while the swarm backoff component provides the delay to perform a new block request.

A block request has five possible outcomes: 1) refusal – the request is refused by the contacted peer, 2) cancellation – the request is canceled by the requester (duplicate block), 3) acceptance – the request is accepted and a block is downloaded, 4) interruption – the request is accepted but the download is interrupted, and 5) disposal – no request is sent due to the lack of eligible peers. Refusal and disposal reveal no information, but all the others do. Cancellation and acceptance reveal that both peers already own that block; interruption reveals that the contacted peer owns that block.

To validate our model, we considered that the peer backoff component is a function of the block transfer time, b_{tt}, and is defined as $\min(\alpha \cdot \lambda^{\tau}, \mu) = \min\left(\frac{b_{tt}}{8} \cdot 2^{\tau}, \frac{k \cdot b_{tt}}{4}\right)$, where α is the peer base backoff time, λ is the exponential factor, τ is the number of consecutive failed requests (all but disposal), and μ is the maximum peer backoff time (25% of download time). The swarm backoff component should be a function of the swarm dynamics to find the proper amount of block requests but, for the sake of simplicity, it is defined as $\beta + \gamma \cdot \tau = 100 + 100 \cdot \tau$, where β is the swarm base backoff time, γ is the scale factor, and τ is the number of consecutive failed request attempts (including disposal).

F. Peer Selection

The peer selection mechanism also helps to maximize the amount of useful blocks of a given content that can be obtained from the available block requests, in the shortest time frame possible, by selecting the peers that return useful blocks in less time. It depends both on the privacy enforcement and on the request backoff mechanisms. The former provides, for a given content, the list of peers to which no further block requests can be sent; the latter provides, also for a given content, which peers are ineligible for the moment, and when can the next block request be performed.

For the validation, given that we considered homogeneous peers and no parallel requests, and also for the sake of simplicity, we select peers randomly.

VI. Validation

This section details the simulation setup, the peer arrival traces used as simulation input, and the use cases considered to validate the Mistrustful P2P model. Given that simulations are

only as good as their models, the simulations were carried out using the ns-3 network simulator [2], which provides realistic network stack and its protocols. Still, the simulation of large scale P2P networks using accurate and realistic models is a complex task. Thereby, to be able to simulate P2P file sharing with thousands of peers using accurate network models, we also use CIDRarchy module [8], a module that we developed for ns-3 that performs IP packet forwarding in constant time.

The validation of our model is done by asserting that peers are able to download contents in due time without advertising what they download. To do so, we simulate the content sharing to evaluate the rate of peers that are able to complete their downloads, and the average download time. Given that the content download due time is subjective, we consider that a content is received in due time if the average download time is, at most, one order of magnitude above direct download time. For the sake of clarity, although cover downloads are required to protect user's content interests, we consider a single content download and no cover downloads. Also, peers are provided with a list of all peers currently in the swarm, request one block at a time, and accept one request at a time. We consider the worst case scenario for how long peers share a given content: peers leave immediately after finishing the download. It is out of the scope of this work to provide a performance comparison with state-of-the-art privacy-enhancing systems.

A. Simulation Setup

We consider a star network topology with a central node mimicking an ISP, and with homogeneous leaf nodes connecting to it through asymmetric links: 30 Mbit/s downlink and 3 Mbit/s uplink. As described in Section V, our model was instantiated as follows. We considered Storm erasure codes, and therefore, any subset of k blocks enables to fully download a content, *seeders* generate a new block per request, and peers only have access to the content after fully downloading it. The privacy mechanism ensures that peers do not exchange more than m blocks with any set of c peers. The block selection, and peer selection mechanisms select, respectively, blocks and peers randomly. The request backoff mechanism sets the peer backoff component as a function of block transfer time that grows exponentially with failed requests, while the swarm backoff component is set as a linear function of failed request attempts.

To ensure that the peer arrival models are realistic, we gathered peer arrival traces of several contents and use them as input to the simulation. The traces were collected by querying a tracker for typical BitTorrent contents, and provide the number of new peers that arrived within ten minute intervals since content publication up to 21 days. We consider the peer arrivals to be independent within each interval, and therefore, we use an exponential function to generate the peer inter-arrival times within that period (Poisson process). We classify content's popularity according to their average peer arrival rate: more popular contents are those that have higher average peer arrival rates. From those collected traces, we selected three video traces for comparison that have different degrees of popularity.

TABLE I
OVERALL NUMBER AND RATIO OF DOWNLOADS COMPLETE.

Contents	1 *Seeder*, Col. of 1	1 *Seeder*, Col. of 32	64 *Seeders*, Col. of 1	64 *Seeders*, Col. of 32
VideoMP_100	75296 (99.82%)	75294 (99.82%)	75323 (99.86%)	75321 (99.85%)
VideoMP_800	74509 (98.78%)	74520 (98.79%)	74565 (98.85%)	74569 (98.86%)
VideoP_100	22444 (99.45%)	22439 (99.42%)	22471 (99.57%)	22472 (99.57%)
VideoP_800	21772 (96.46%)	21762 (96.42%)	21808 (96.63%)	21821 (96.69%)
VideoLP_100	3308 (99.91%)	3257 (98.37%)	3308 (99.91%)	3308 (99.91%)
VideoLP_800	3257 (98.37%)	3240 (97.86%)	3271 (98.79%)	3269 (98.73%)

B. Use Cases

For each individual peer arrival trace we consider eight use cases, which are a result of combining three distinct variables, each taking one of two possible values. We consider a privacy protection level against single peer attacks (collusion of 1) or collusion group attacks of, at most, 32 peers (collusion of 32). Contents are always divided into 64 blocks, have a size of either 100 MiB or 800 MiB, and are shared either by 1 *seeder* or 64 *seeders*; *seeders* are always present during the content sharing. Given that, for the traces we collected, the peer arrival peak usually occurs within the first 36 hours, each use case is simulated for 48 hours to encompass, at least, the content bootstrap and the content sharing peak. We consider $m = k - 1$, i.e., no single peer can download all k blocks from peers belonging to a group of c peers.

Our goal is to validate the model for different content popularities, privacy protection levels, content sizes, and number of *seeders*.

VII. RESULTS AND DISCUSSION

In this section, we present the simulation results for the validation of the Mistrustful P2P model. For each use case, we measured the rate of peers that completed the download, and the average download time. All values are for one hour intervals, thus, for the sake of clarity, we use 'overall' to differentiate between the values for the whole simulation and those for one hour intervals. Figure 1 depicts the number of downloads completed over time, and Table I provides the overall number of downloads completed and the overall completion rate. The average download time is illustrated on Figure 2 while the average overall download time, and the ratio to the direct download time are presented on Table II. Content download is limited by the uplink (3 Mbit/s) because block requests are performed one at a time, thus to a single peer at a time. As so, the reference download times for direct download of 100 MiB and 800 MiB contents are, respectively, approximately 5 and 38 minutes.

The results demonstrate that our model is feasible as peers were able to complete their downloads, and do so in due

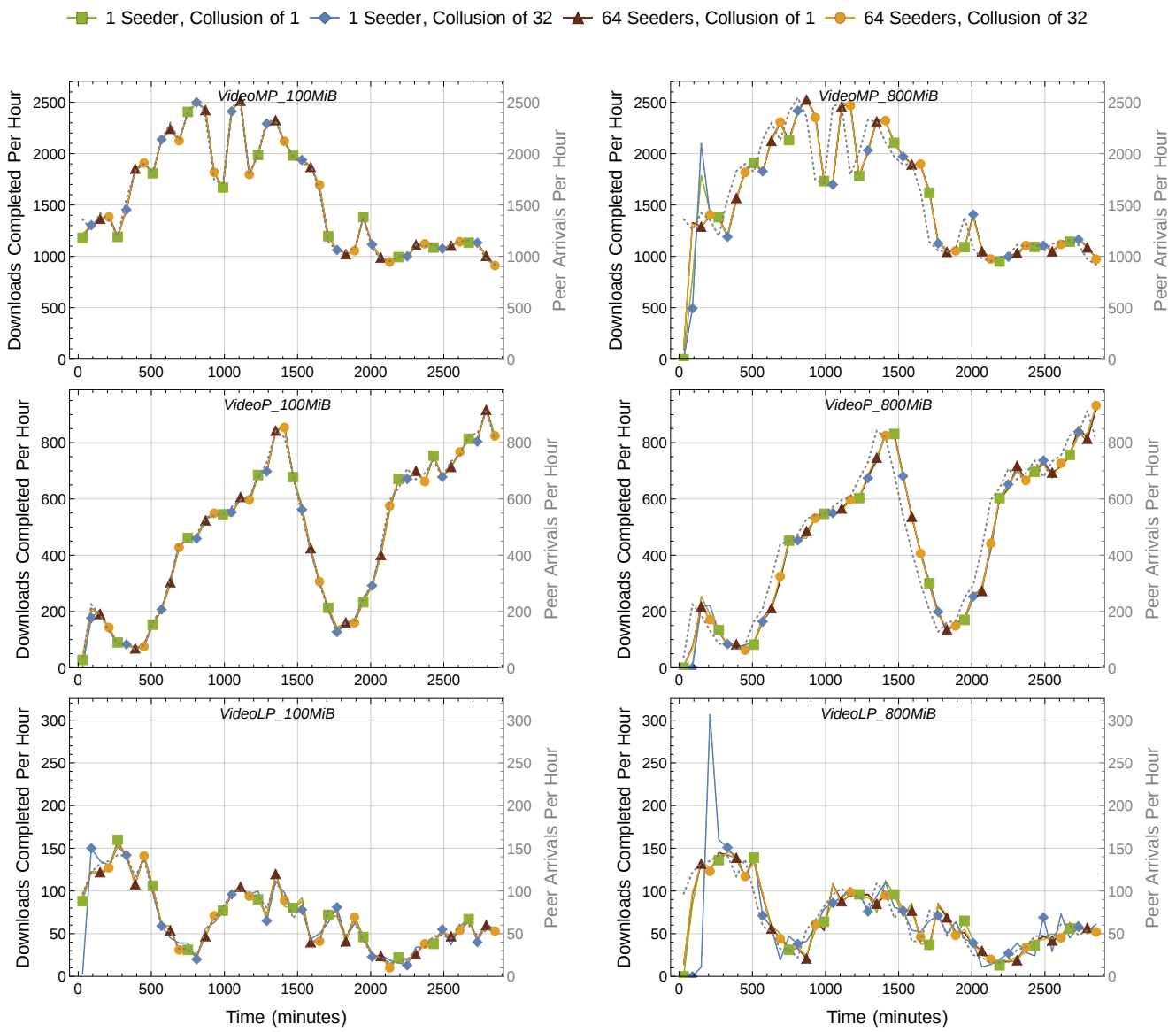

Fig. 1. Number of downloads completed over one hour periods for 100 MiB (left) and 800 MiB (right) contents using a more popular (*MP*), a popular (*P*), and a less popular (*LP*) peer arrival traces as input (one per row). Each plot depicts four use cases that are a result of using either 1 or 64 *seeders*, and considering either single peer attacks or collusion attacks of, at most, 32 peers. The peer arrival rate is represented in gray with a y-scale on the right.

time, without advertising what they have downloaded. For most use cases, the average overall download time is close to the direct download time (see Table II). As shown in Figure 1, peers download completion rate closely follows the peer arrival rate with an offset, which increases as the size of the content increases because peers need to stay longer to fully download the content. Table I shows that the overall download completion rate is very high; the only peers that have not completed the download are those that were sharing when the simulation stopped. Figure 2 shows that peers complete their downloads in due time, and that the average download time depends on the peer arrival rate (content popularity), the privacy protection level, the number of *seeders* sharing the content, and on the content size.

The average download time decreases down to a minimum near the direct download time as the peer arrival rate increases. As the peer arrival rate decreases, both the average download time and the download time variance increase, which suggests that some peers have to wait for others to join before being

TABLE II
AVERAGE OVERALL DOWNLOAD TIME, IN MINUTES, AND RATIO TO DIRECT DOWNLOAD TIME.

Contents	**1 *Seeder*, Col. of 1**	**1 *Seeder*, Col. of 32**	**64 *Seeders*, Col. of 1**	**64 *Seeders*, Col. of 32**
VideoMP_100	8.4 (1.8)	8.4 (1.8)	6.8 (1.5)	6.8 (1.5)
VideoMP_800	60.7 (1.6)	60.8 (1.6)	57.8 (1.6)	57.6 (1.5)
VideoP_100	8.4 (1.8)	11.0 (2.4)	6.0 (1.3)	6.0 (1.3)
VideoP_800	60.1 (1.6)	60.5 (1.6)	55.5 (1.5)	55.4 (1.5)
VideoLP_100	8.4 (1.8)	51.1 (11.0)	5.2 (1.1)	5.2 (1.1)
VideoLP_800	61.0 (1.6)	85.1 (2.3)	47.8 (1.3)	48.0 (1.3)

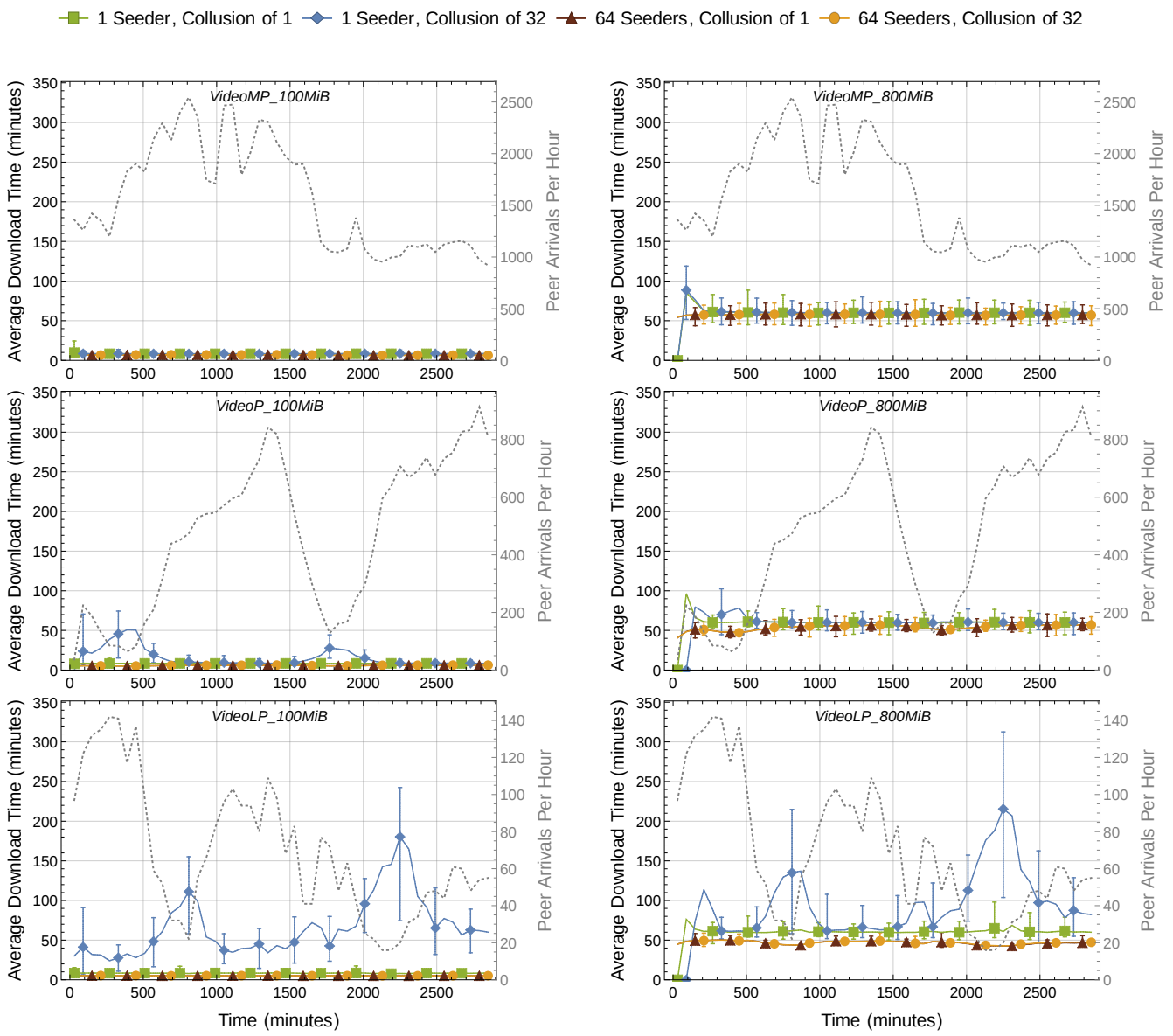

Fig. 2. Average download time over one hour periods for 100 MiB (left) and 800 MiB (right) contents using a more popular (*MP*), a popular (*P*), and a less popular (*LP*) peer arrival traces as input (one per row). Each plot depicts four use cases that are a result of using either 1 or 64 *seeders*, and considering either single peer attacks or collusion attacks of, at most, 32 peers. The bars represent the minimum and maximum download times within one hour intervals. The peer arrival rate is represented in gray with a y-scale on the right.

able to complete the download. The numbers of peers that need to be contacted is constrained by the privacy protection level (at least $c+1$ peers) but also depends on how successful the block requests are, which in turn are dependent on other variables such as the block distribution among the peers. Therefore, the results suggest that the number of peers that need to be contacted is higher than that imposed by the privacy protection level ($c+1$), and the average download time increases when those peers are not immediately available. Adding *seeders* provides a two-fold improvement on the average download time: 1) since *seeders* are always present, less *commoners* need to be simultaneously sharing to be able to complete the download; 2) *seeders* improve the probability of successful block requests as they always offer a useful block, which increases the number of unique blocks available on the network. Unlike direct download time, the average download time does not increase linearly with the increase of the content size. The average number of peers present in the network increases with the increase of the content size because *commoners* have to stay longer to fully download the content, therefore increasing the probability of successful requests, which, in turn, contributes to a lower average download time. This is more evident for less popular contents: for the less popular peer arrival trace, despite the 800 MiB content being eight times larger than the 100 MiB one, the average download time is only less than two times higher.

In sum, the results demonstrate that our model is feasible and, for most of the use cases considered, the average overall download time is close to the direct download time. We considered an instantiation of our model that focus on simplicity instead of optimality, and the peer download completion rate is still very high. For the 64 *seeders* use cases, the average download time is very close to the direct download time, even for a privacy protection level against collusion group attacks of, at most, 32 peers.

VIII. Conclusions

We proposed a novel P2P file sharing model that provides deterministic protection of user's content interests, against attacks of size up to a configured privacy protection level, by avoiding the advertisement of what peers download, as long as the effective size of the largest colluding group does not exceed the one configured; it supports open and untrustworthy P2P networks, and requires no trust links between peers. Our model thwarts passive attacks differentiating genuine from cover downloads using solely block advertisements, and forces attackers to engage in content sharing to know which blocks a peer owns.

By avoiding block advertisement, our model enables peers to control individually what information is disclosed to other peers, and has no requirements on the amount of blocks that have to be downloaded per cover content, so that no single colluding group is able to identify it as a cover content. As so, novel disguise schemes can be devised to conceal user's interests that use more cover contents without increasing the network overhead.

We demonstrated its feasibility through simulation, using ns-3, considering an instantiation of our model focused on simplicity rather than on optimality, and where peers leave immediately after finishing the download. In the majority of the use cases considered, the average overall download time is close to the direct download time. With the Mistrustful P2P model, peers have no need to suddenly terminate or remove downloads because the privacy protection level does not depend on the time a peer keeps sharing a content, as long as cover and genuine downloads are treated in the same way.

As future work, we intend to (1) compare our model against a simple traditional P2P file sharing model, (2) improve the request backoff mechanism to increase the overall performance, and (3) conduct further experiments to evaluate the Mistrustful P2P model with more peer arrival traces, mainly less popular ones, and include more variables such as the number of blocks into which a content is divided. Then, we will analyze how the probability of successful requests changes over time, so that we can improve the instantiation of our model herein presented. We will also propose cover download selection algorithms that minimize the amount of cover traffic required while preserving the privacy protection.

Acknowledgment

This work is financed by the ERDF – European Regional Development Fund – through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme – within project "POCI-01-0145-FEDER-006961", and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013 and under the fellowship SFRH/BD/69388/2010.

References

[1] Freenet project. https://freenetproject.org/.
[2] ns-3 network simulator. https://www.nsnam.org/.
[3] K. Bauer, D. McCoy, D. Grunwald, and D. Sicker. BitBlender: Light-weight anonymity for BitTorrent. In *Proceedings of the Workshop on Applications of Private and Anonymous Communications (AlPACa 2008)*. ACM, September 2008.
[4] A. Chaabane, P. Manils, and M. Kaafar. Digging into anonymous traffic: A deep analysis of the Tor anonymizing network. In *Network and System Security (NSS), 4^{th} International Conference on*, pages 167–174, 2010.
[5] S. Chakravarty, G. Portokalidis, M. Polychronakis, and A. Keromytis. Detection and analysis of eavesdropping in anonymous communication networks. *International Journal of Information Security*, 14(3):205–220, 2015.
[6] D. R. Choffnes, J. Duch, D. Malmgren, R. Guiermà, F. E. Bustamante, and L. Amaral. SwarmScreen: Privacy through plausible deniability in P2P systems. Technical report, Northwestern EECS, March 2009.
[7] R. Cuevas, M. Kryczka, A. Cuevas, S. Kaune, C. Guerrero, and R. Rejaie. Is content publishing in BitTorrent altruistic or profit-driven? In *Proceedings of the 6^{th} International COnference*, Co-NEXT '10, pages 11:1–11:12, New York, NY, USA, 2010. ACM.
[8] P. M. da Silva, J. Dias, and M. Ricardo. CIDRarchy: CIDR-based ns-3 routing protocol for large scale network simulation. In *Proceedings of the 8^{th} International Conference on Simulation Tools and Techniques*, SIMUTools '15, pages 267–272, 2015.
[9] P. M. da Silva, J. Dias, and M. Ricardo. Storm: Rateless MDS erasure codes. In *Wireless Internet*, pages 153–158. Springer, 2015.
[10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13^{th} Conference on USENIX Security Symposium - Volume 13*, SSYM'04, 2004.
[11] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, Feb. 1999.
[12] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving P2P data sharing with OneSwarm. *SIGCOMM Comput. Commun. Rev.*, 41(4):–, Aug. 2010.
[13] S. Katti, J. Cohen, and D. Katabi. Information slicing: Anonymity using unreliable overlays. In *Proceedings of the 4^{th} USENIX Conference on Networked Systems Design and Implementation*, NSDI'07, 2007.
[14] S. Le Blond, P. Manils, A. Chaabane, M. A. Kaafar, C. Castelluccia, A. Legout, and W. Dabbous. One bad apple spoils the bunch: Exploiting P2P applications to trace and profile Tor users. In *Proceedings of the 4^{th} USENIX Conference on Large-scale Exploits and Emergent Threats*, LEET'11, 2011.
[15] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, PODC '02, pages 233–242. ACM, 2002.
[16] S. J. Nielson and D. S. Wallach. The BitTorrent anonymity marketplace. *CoRR*, abs/1108.2718, 2011.
[17] R. Petrocco, M. Capotă, J. Pouwelse, and D. H. Epema. Hiding user content interest while preserving P2P performance. In *Proceedings of the 29^{th} Annual ACM Symposium on Applied Computing*, pages 501–508. ACM, 2014.
[18] M.-Z. Shieh, S.-C. Tsai, and M.-C. Yang. On the inapproximability of maximum intersection problems. *Information Processing Letters*, 112(19):723 – 727, 2012.
[19] P. Tsang, A. Kapadia, C. Cornelius, and S. Smith. Nymble: Blocking misbehaving users in anonymizing networks. *Dependable and Secure Computing, IEEE Transactions on*, 8(2):256–269, March 2011.