Asynchronous Covert Communication Using BitTorrent Trackers

Mathieu Cunche, Mohamed Ali Kaafar, Roksana Boreli

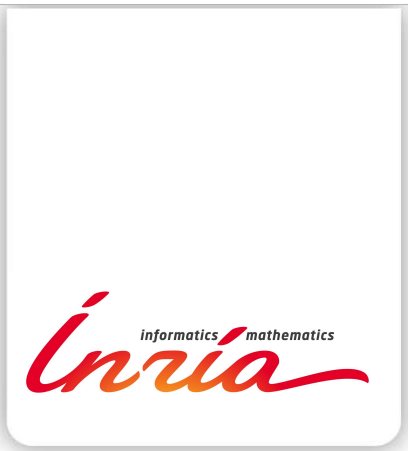
Asynchronous Covert Communication Using BitTorrent Trackers

Mathieu Cunche[*†], Mohamed Ali Kaafar[†‡], Roksana Boreli[‡]

[†] **Inria, PRIVATICS, France**
[*] **INSA-Lyon, CITI, France**
[‡] **National ICT Australia**

RESEARCH REPORT
N° 8554
June 2014
Project-Team PRIVATICS

ISSN 0249-6399 ISRN INRIA/RR--8554--FR+ENG



Asynchronous Covert Communication Using BitTorrent Trackers

Mathieu Cunche*†, Mohamed Ali Kaafar†‡, Roksana Boreli‡

† Inria, PRIVATICS, France
* INSA-Lyon, CITI, France
‡ National ICT Australia

Project-Team PRIVATICS

Research Report n° 8554 — June 2014 — 18 pages

Abstract: Covert channels enable communicating parties to exchange messages without being detected by an external observer. In this paper we propose a novel covert channel mechanism based on BitTorrent trackers. The proposed mechanism uses common HTTP commands, thus having the appearance of genuine web traffic and consists of communications that are both indirect and asynchronous: no messages are directly exchanged between the sender and the receiver (of covert communications) and there is a potentially considerable delay between the sender's message to the relaying party and the receiver collecting this message. We present details of the proposed scheme in which a centralized BitTorrent tracker is used for storing covert messages and evaluate its performance based on the implemented prototype. We analyze the detectability of covert communications by an adversary and show that, while the common nature of the BitTorrent traffic and the large number of clients make the detection unlikely, the low temporal correlation between the writer and the reader (the two communicating parties) further increases the detection difficulty. Finally we discuss a variant of our scheme that uses a decentralized tracker (based on distributed hash tables), increasing the scalability and enabling a larger number of parallel covert communication channels.

Key-words: Covert Channel, BitTorrent, Tracker

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Canaux de Communication Cachés et Asynchrones via les Trackers BitTorrent

Résumé : Les canaux de communication cachés permettent à des parties d'échanger des messages sans être détectés par un observateur extérieur. Dans ce papier, nous proposons un nouveau mécanisme de canaux cachés reposant sur les trackers BitTorrent. Le système proposé repose sur l'utilisation de commandes HTTP, lui donnant ainsi l'aspect d'un simple trafic web. De plus ce système permet des communications qui sont à la fois directes et asynchrones : aucun message n'est directement échangé entre l'émetteur et le récepteur (du canal caché) et il peut exister un délai considérable entre le moment où le message est transmis au relai et celui où il est lu. Nous présentons les détails du système proposé dans lequel un trackeur BitTorrent est utilisé pour stocker les messages, et nous évaluons ses performances grâce à un prototype. Nous analysons la détectabilité des communications cachés par un adversaire, et nous montrons qu'en plus de la nature commune des communications BitTorrent et du grand nombre d'utilisateurs de ce système, la faible corrélation temporelle entre les opérations de lecture et d'écriture rendent la détection difficile. Finalement, nous présentons une variante de ce système utilisant un trackeur décentralisé (basé sur une table de hachage distribuée), qui permet un meilleur passage à l'échelle et l'utilisation de plusieurs canaux en parallèle.

Mots-clés : Canaux Cachés, BitTorrent, Tracker

1 Introduction

While the purpose of encryption is to protect the content of any communication from being revealed to unauthorized parties, covert channels aim to hide the existence of the communication itself. Covert channels were first described in the context of Multi-Level Security (MLS) systems [3], as a way to communicate information from high security to low security processes. They have been subsequently extended to the case of network communications, with their application ranging from military and national security, where they are used to hide potential communications between parties, to botnets and use by cyber criminals, that look to control remote systems while circumventing traffic filtering.

The model of covert channels has been initially introduced by Simmons in [12] with the Prisoners' problem and the subliminal channel. Two prisoners, who communicate while being observed by a warden, exchange seemingly harmless messages, which however contain useful information enabling them to agree on an escape plan.

We define as indirect network covert channel (I-NCC) a covert channel in which the information is being sent to a host that is distinct from the participant end systems, while in a direct network covert channel (D-NCC) the information is stored in the data packets exchanged between the participants. A large number of dynamic D-NCCs have been proposed in the past (e.g. [14, 6]). In those systems, the transmitted information is embedded within various fields of network packet headers, or inside the data used for padding. On the other hand a fewer number of A-NCCs have been proposed. The utilized techniques include e.g. [7] where the counters available on various websites are leveraged to store information.

The huge amount of data exchanged through Peer-to-Peer (P2P) networks and the global-scale of P2P infrastructure creates an environment where communication has a potential to be hidden among the mass of regular users and corresponding host message exchanges. We propose to use the BitTorrent tracking infrastructure, as the most widely used P2P file-sharing system with tens of millions of worldwide users, to build a novel asynchronous (network storage) covert channel.

The choice of the BitTorrent tracker system as a storage for our covert channel algorithm is motivated by the suitable properties of the tracker system. First, a tracker commonly hosts lists of peers for multiple content files, and many trackers allow users to not only add content to a swarm but also to retrieve the complete list of content "monitored" by a tracker. Second, trackers usually scale to millions of requests from hosts geographically distributed across many countries, which make them good candidates for universally accessible memory.

In this paper we present a novel asynchronous covert channel based on BitTorrent trackers. Our contributions are as follows.

We present a communication scheme that enables two parties to perform a hidden exchange of information through the centralized BitTorrent tracker. Two inherent properties of the scheme are beneficial to covert communications, as they assist with having a low detectability: the scheme is an indirect network covert channel and it is asynchronous, with the sending and retrieving nodes having the flexibility of not needing to access the intermediate node at the same time. The scheme includes error correction and encryption mechanisms, respectively, to mitigate data losses and to provide obfuscation of transmitted data. Our system exploits the basic primitives of the (centralized) BitTorrent tracker protocol, and therefore does not require the cooperation of the tracker.

We implement a prototype and evaluate the performance of the proposed covert channel scheme. The presented results show that our system can transmit data at an average rate of 1.89 bits/second, and that the data losses appearing on the tracker are compensated by the error correction scheme. In addition, once written on the tracker, the data remain available for

dozens of minutes, providing a considerable time window in which the receiver may retrieve the data. Overall these results demonstrate that such a communication channel can reliably transmit information at a rate sufficient for the exchange of small pieces of data (such as text messages or botnet commands) without a fine-grained time synchronization between the sender and the receiver.

We analyze the detectability of our scheme, which includes a number of features well suited to covert communications. First, our system is solely based on one of the most commonly used operation on the Internet, i.e. the HTTP/GET request, resulting in communications having the appearance of genuine traffic. Second, it is asymmetric in the sense that read and write operations are not equivalent, resulting in a distinct cost and a detection difficulty for the sender and the receiver.

Finally, we propose an extension of our scheme to include a decentralized tracker, based on distributed hash tables (DHT), a mechanism adopted by multiple BitTorrent clients. We describe the adaptation of our covert communication scheme to the DHTs and discuss it's advantages of increased scalability and limitations of an increased potential for traffic detection, compared to the scheme where centralized trackers are used.

The paper is organized as follows. In Section 2 we briefly overview the BitTorrent system. Section 3 describes the design of our covert channel and Section 4 analyses the characteristics of the centralized tracker based system. In Section 6 we discuss the difficulty of detection. In Section 5 we present results on the system's performance. Modification that includes the use of distributed hash table is presented in Section 7 and Section 8 review the related work. We present conclusions in Section 9.

2 Centralized BitTorrent Trackers

BitTorrent is a Peer-to-Peer system in which peers share and download content by exchanging content chunks. To establish a connection with other peers, any peer needs to obtain the IP addresses and ports of the peers involved in the swarm (set of peers downloading and/or sharing a given content), which is performed by using a central server *tracking* the peers involved in a swarm. Each centralized tracker usually stores the list of peers for several contents. For instance, we found several millions of contents registered on the *publicbt* tracker.

The BitTorrent system uses a content identification system based on hash called *infohash* obtained by computing the SHA-1 digest of the content meta-information. Therefore an infohash is a 20 bytes long (160 bits) identifier. In order to obtain the list of peers connected to a swarm, a BitTorrent client contacts a centralized tracker using an *Announce* request. This request can be performed through either UDP or HTTP protocol. In both cases the requested URL contains several parameters, in particular the content's identifier : `http://[tracker_url]/announce?info_hash=x`.

Upon reception of an announce request, and if the content is already registered on the tracker, the tracker returns a set of peers and adds the requesting peer to the list. If the content is not registered on the tracker, there are two possible cases: either the tracker rejects the request by specifying a request for an unregistered content, or the tracker registers the content and adds the requesting peer to the corresponding peer list. In the later case the tracker is called *open*, in the sense that any peer can register a new content on the tracker by performing an announce request. Within the BitTorrent ecosystem, both open and non-publicly accessible centralized trackers co-exist.

Additionally, centralized BitTorrent trackers support other requests such as *scrape* requests, used to retrieve information on a given content such as the number of registered peers. A

generalisation of this request is the *scrape-all* request that provides the same information but for all the contents registered on the tracker. As a side effect, this request also provides the full list of the contents registered on the tracker. A *scrape-all* request is performed through an HTTP/GET query on the following url : `http://[tracker_url]/scrape`

Since, the *scrape-all* request can be very costly for the tracker, it is not always supported. In order to remove this burden from the tracker itself, the corresponding information is sometime offered by a web server. For instance, the *publicbt* tracker provides a compressed scrape file refreshed every 10 minutes at the following address : `http://publicbt.com/all.txt.bz2`.

3 Centralized Tracker-Based Covert Channel Design

In this section, we outline the design of the proposed asynchronous cover channel.

3.1 Read and Write operations

We first describe how the centralized tracker can be used as a memory and how the read and write operations can be performed. In this memory the information is stored under the form of data block represented by *info_hash* identifier in the tracker's content table. Since the *info_hash* used as content identifier is 20 bytes long, the information can be stored on the tracker as chunk of 20 bytes (160 bits).

Fig. 1 depicts the communication between the writer and the tracker on one side, and between the reader and the tracker on the other side. To write on the tracker, we have to add an entry using an announce request. Let x be a 20 bytes length value, then sending the following request to the tracker will add an entry x to the content table: `GET http://[tracker_url]/announce?info_hash=x`.

It is possible that the entry x already exists in the content table, in which case, the entry will be updated by increasing the number of peers and adding the source IP address of the request to the list of connected peers.

To read a value from the tracker, we have to get the list of the content identifier stored on the tracker. This operations can be done by using the *scrape all* request on the tracker. As noted in Section 2, some tracker, such as `publicbt.com`, provide this information under the form of an aggregated file (we used the compressed text file provided by `publicbt.com` that can be retrieved at `http://publicbt.com/all.txt.bz2`). Some tracker provide this information under the form of an aggregated file. For instance `publicbt.com` provides a link to a compressed text file that can be retrieved at `http://publicbt.com/all.txt.bz2`.

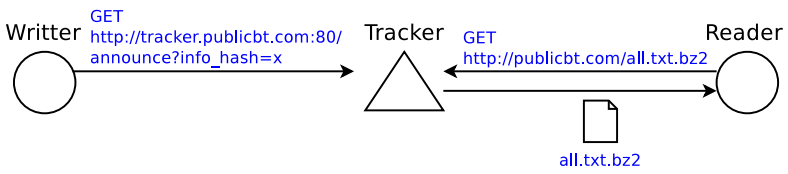


Figure 1: Read and write operation on the tracker.

3.2 Hash-based filtering for packet identifications

The previously described read operation retrieve all the *info_hash* stored in the content table. This content table contains both genuine entries, i.e. entries corresponding to existing torrent,

and *hidden* entries, i.e. entries added to the tracker by our hidden communications. To distinguish *hidden* entries from *genuine* entries, we use a hash-based filtering mechanism. For each piece of information stored on the tracker, the last t bits are a hash value of the $160 - t$ first bits, using a hash function, $h(.)$. For the read operation, after the *scrape all* operation has been performed, the retrieved *info_hash* are filtered. Specifically, for each *info_hash* x, if $h(x[0 \ \ldots \ 160 - t - 1]) = x[160 - t \ \ldots \ 160 - 1]$, then keep x otherwise discard x.

This hash-based filtering approach may however induce false positives, where *info_hash* may pass the filter while they are genuine and not hidden entries. We discuss this issue in Section 4.1.

3.3 Obfuscation through encryption

With the previous hash-based filtering mechanism, the *hidden info_hash* can be easily detected by someone knowing the algorithm. This issue can be avoided by encrypting the data before it is written on the tracker, for instance with a symetric block-cipher. Indeed, only a person knowing the secret key will be able to decrypt the retrieved *info_hash* and filter out those that are corresponding to hidden pieces of information. In addition, this encryption provide data confidentiality and in particular prevent an eavesdropper to recognize the hidden data in plain text.

3.4 Data indexing

The size of the *info_hash* limits the amount of data that can be transmitted in one entry of the content table. The quantity of information that can be encoded (written) is even more reduced with the adjunction of the hash of the useful data (as described in Section 3.2). In essence, a total of $160 - t$ bits of information can transmitted per entry. In the case where a larger amount of data has to be transmitted, the data can be split and carried by several content identifiers. In order to transmit a message split in several pieces, we need to add an indexing functionality. To this aim we reserve the first s bits of data for indexing. Those s bits simply contain a sequence identifier, and are concatenated to the data before the addition of the hash to allow the indexing of up to 2^s pieces of information.

3.5 Data loss and erasure coding

During our experiment, we have observed that the data written on the tracker can be subject to erasure[1], i.e. loss of information. These data losses are caused by various errors and by the content expiry delay (which is studied in Section 4.2).

Erasure coding provides an elegant and efficient solution to the problem of data losses encountered in storage [9] and computer communications applications [11]. By adding redundancy to the source data, erasure codes allow to tolerate the loss of data blocks. More formally the encoding process takes as input a set of k data units and produces n encoded data units, where $n > k$. Perfect codes, allow the source data to be rebuild from any m distinct data units, as long as $m \geq k$. Reed-Solomon codes [10] are an example of perfect codes that have been extensively used in the context of data erasure.

After splitting the message into k blocks of $160 - t - s$ bits, the encoding process is applied and n blocks, also of $160 - t - s$ bits, are generated as output of the encoder. Thanks to the optimal erasure correction capabilities of the Reed-Solomon codes, the reader can recover the source data as soon as k blocks have been collected. The amount of redundancy represented by

[1] Data written on the tracker was not returned in the *scrape-all* results.

the code rate, $R = n/k$, can be adjusted to meet the channel requirements, i.e. as appropriate to the loss rate.

3.6 Summary of the covert channel mechanism

On the writer side, the message is first divided into k blocks of $160 - t - s$ bits and encoded into n blocks. For each block, an index and a hash are then added and the obtained data is encrypted before being written (through a HTTP Get Request) on the tracker. Figure 3 describes the encoding of a single data block while Figure 2 presents the operations required to encode a message spanning over several data blocks.

To retrieve the message, the reader performs a *scrape-all* request, then decrypts all the content identifiers and uses the hash-based filter to select the hidden data blocks. If enough blocks have been collected, a decoding process is applied to decode the original message.

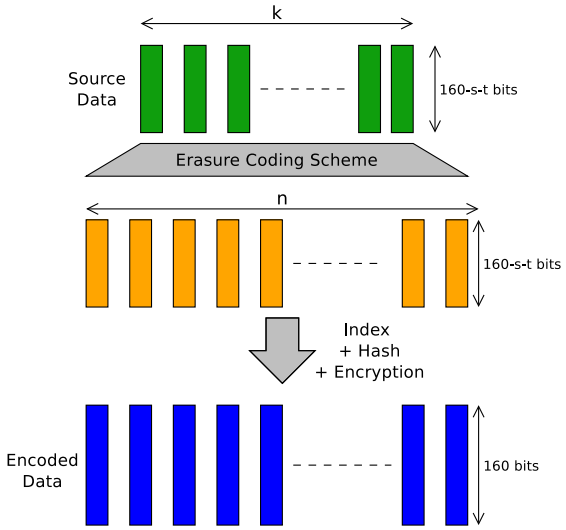


Figure 2: Encoding of a message spanning over several blocks.

4 System analysis

In the following, we analyze the characteristics of our proposed asynchronous covert communication scheme. First, as previously mentioned, we examine the extent to which the hash-based filtering approach may induce false positives while retrieving the data from the centralized tracker. Then we analyze the time data blocks remain on the tracker and finally we evaluate the fraction of data block that are lost during the transmission.

4.1 False-positive of the hash-based filter

As noted in Section 3.2, identification of the hidden content by a hash function-based filter may induce false positives, when genuine content identifiers successfully pass the hash-based filter. A false positive will result in either a block collision (if the block index already exists in the sequence) or an insertion (if the index does not exist in the sequence). In both cases the false positive will disrupt the communication by corrupting the data.

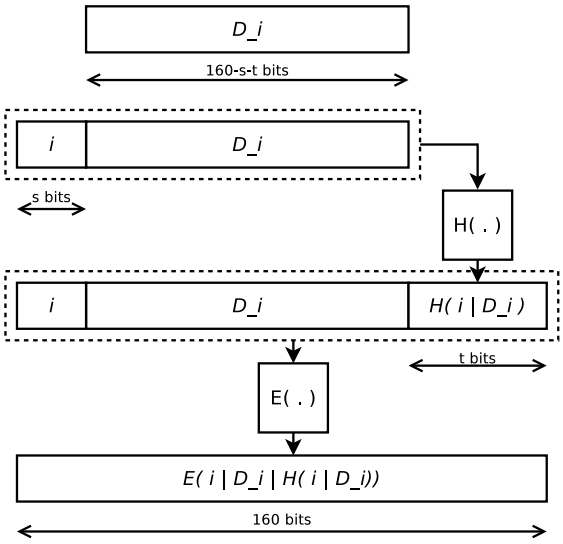


Figure 3: Creation of the forged content identifier.

Let $(X \mid Y)$ be the content identifier where X represents the $160-t$ first bits and Y represents the t last bits corresponding to the hash. A false positive is a genuine content identifier $(X \mid Y)$ such as $h(X) = Y$, ie a content identifier for which the last t bits match the hash of the $160 - t$ bits. A content identifier being 20 bytes long SHA-1 hash, and assuming that the output of SHA-1 is uniformly distributed, the false-positive probability is : $P(h(X) = Y) = 2^{-t}$. For a table containing n *genuine* contents, the expected number of false positive is: $N_{\text{F-P}} = n\ 2^{-t}$

The expected number of false positives can therefore be adjusted through the hash size, t. The number of false positives should be kept as small as possible. However, increasing the size of the hash reduces the number of useful bits in the data block. Let's exemplify this by considering the *publicbt* tracker which is tracking around 3.10^6 content identifiers. With this tracker, a 2-bytes long hash ($t = 16$) yields to 43.7 false positives, while a 4-bytes long hash ($t = 32$) yields to 6.67 10^{-5} false positives on the average. We then consider the latter as a good compromise between communication overhead induced by the hash size and the expected number of transmission failure caused by false positives.

4.2 Entry's lifetime

Entries are not stored indefinitely on the centralized tracker and if an entry is not requested for a given amount of time, it is removed from the tracker. We have measured this expiration delay on the *publicbt* tracker, adding several content identifiers to the tracker, and then performing a scrape-all every 10 minutes[2].

Fig. 4 presents a cumulative distribution function of the expiry time on the *publicbt* tracker. We observe that entries remain valid on the tracker for an average period of 2823 seconds (∼47 minutes), with a minimum of 2275 seconds (∼38 minutes). This large delay allows asynchronous communications, since the message will remain available on the tracker for dozens of minutes.

[2]This value has been chosen to limit the load on the tracker on the one side, and because the scrape-all file is refreshed every 10 minutes

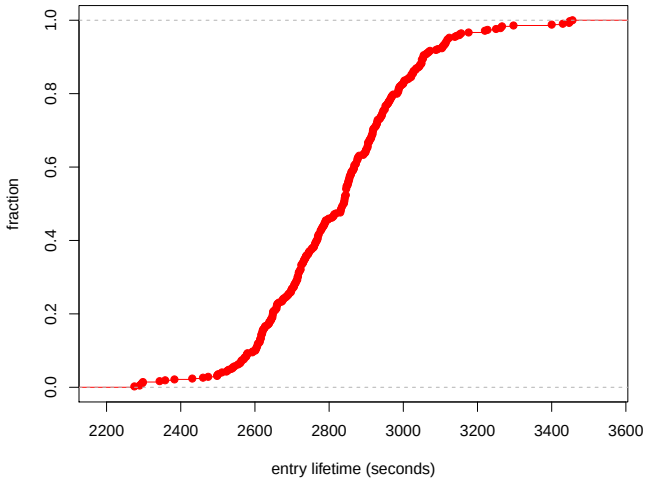


Figure 4: Expiration delay distribution.

4.3 Data loss

As noted in Section 3, some data blocks can be lost in addition to the erasures caused by the previously discussed expiration delay. Indeed, it may happen that a part of the data blocks written on the tracker are never found in the identifier lists returned by the *scrape-all*. By performing a number of write operation on the tracker while periodically (every 10 minutes) performing scrape-all operations, we have estimated the fraction of blocks lost, observing over the set of 10 experiments, an average loss rate of 44.92%. As stated in Section 3.5, data losses can be tolerated with the aid of an erasure coding scheme, for instance, a Reed-Solomon code.

The amount of redundancy (determined by the code rate R) added by the coding scheme must be chosen according to the loss rate of the channel on which the data is transmitted. In theory, to reliably transmit data over a channel of loss rate p, one need to use a coding scheme configured with a code rate $R < 1 - p$. In practice, because the actual loss rate can vary around the computed average, one must take a margin from the theoretical value. In the following, we will consider coding schemes configured with code rates ranging from $R = 1/2$ (close to the theoretical limit of $1 - p = 0.5508$) to $R = 1/5$.

5 Performance Analysis

This section studies the performance of the proposed communication system. In particular, we evaluate the transmission success rate as well as the throughput of write and read operations. We have implemented a prototype comprising a writer and a reader and performed experiments using the centralized tracker `publicbt.com`. We consider a scenario were a unique sender wants to transmit a message to a unique receiver. The system is configured with a hash-size of $t = 32$ bits, and an index composed of $s = 6$ bits, leaving $160 - 6 - 32 = 122$ bits of payload in *info_hash*. A message is divided into k blocks of 122 bits, that are then encoded by the erasure

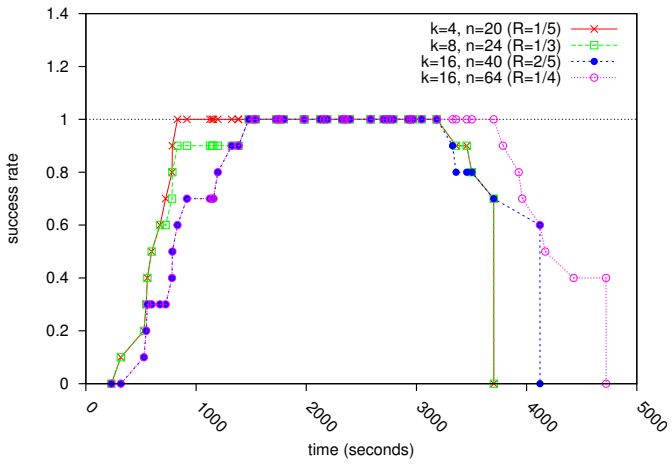


Figure 5: Success rate as a function of the time elapsed since the end of the first write operation.

protection scheme into n block of 122 bits. An index of $s = 6$ bits along with a hash of $t = 32$ bits are happened to each of those n blocks, before being encrypted with a key shared by the sender and the receiver. With those parameters, the expected false positive rate of the hash-based identification is 6.67 10^{-5}, and the maximum number of data block (including the data produced by the erasure protection scheme) is $2^s = 64$.

5.1 Transmission success rate

As explained in Section 4, information written on the centralized tracker can be erased. The Reed-Solomon codes [10] are used to provide resilience against erasures. Several configuration with various message sizes and code rates have been considered. We have considered messages fitting in $2, 4, 8, 16$ data blocks and code rates varying between 1/5 and 1/2.

For each set of parameters (message size and code rate) we perform the write and read operations and observe the success rate of the transmission of the entire message. For the write operations, the data blocks are distributed amongst 5 threads that performs the write operations in parallel. At the same time, read operation are performed at regular interval (every 10 minutes). We then compute the success rate of a read operation as a function of the time elapsed since the end of the first write operation. A read operation is considered successful if the information contained in the retrieved scrape file is sufficient to decode the message. The experiment is performed 10 times for each set of parameters. Note that the success rate depends on the data retrieved from one scrape operation. Also note that the success rate could be further increased by considering the information retrieved over multiple read operations.

The associated results are presented in Fig. 5. We have limited the results shown to the configurations that have provided the best performances. Focusing on the the selected configurations, there is a time frame during which a read operation is always successful. The length of this time frame depends on the parameters, and ranges as depicted on Fig. 5, from 1700 seconds to 2350 seconds, allowing for asynchronous transmission as the message can be read anytime during a time frame of several tenths of minutes.

5.2 Transmission throughput

The proposed hidden channel will have an associated throughput, i.e. the data rate at which the sender will be able to transmit data to the receiver. As all communications are based on elementary operations, we first measure the time required to perform these in a series of 100 experiments. Fig. 6 shows the resulting cumulative distribution of the time duration for elementary write and read operations. We can observe that the read time can take significantly longer than the write operation, with 50% of read attempts having a duration longer than n 450 seconds.

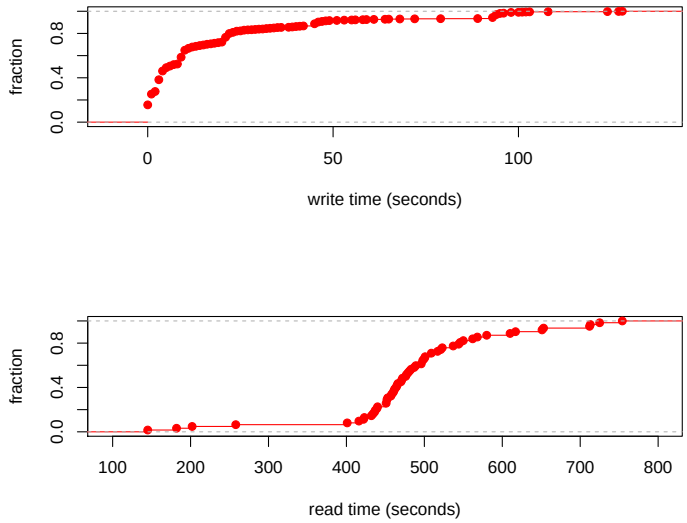


Figure 6: Cumulative distribution of the time required to complete the elementary operations of read and write.

We note that to transmit a message, a number of write operations need to be done (one for each block i.e. elementary message), while only one elementary read operation is required (the *scrape-all* operation). Consequently, the time to write a message will depend both on the message size (number of blocks) and the elementary write delay, while the read operation will result in a constant delay. Assuming a large number of write operations, the throughput can therefore be approximated by the throughput of the write operation.

Table 5.2 presents a summary of the throughput results, for the following parameters: $s = 6$ and $t = 32$, with 122 of useful bits per data block that can be used to transmit information. With the elementary read operation resulting in an average delay of 111.289 seconds and the elementary write operation an average delay of 65.475 seconds, the resulting write throughput (and the system throughput) is 1.89 bits per seconds. Note that performing the write operations in parallel has a potential to increase the system throughput.

Table 1: Summary of the delay introduced by the elementary operations and the resulting throughput.

Operation	Elementary operation time min / avg / max	Average throughput
Write	0 / 65.475 / 532 s	1.89 b/s
Read	86 / 111.289 / 175 s	NA

6 Detectability Analysis

We now discuss the difficulty of detecting our covert-channel based communication. We consider two types of wardens: a weak adversary (monitoring warden) that has no knowledge of the scheme used for communications, and a strong adversary that knows all the parameters of the used scheme, with the exception of the secret key used to encrypt the data, as discussed in section 3.3.

The goal of a warden ignoring the proposed protocol is to spot communication corresponding to our scheme. We will now discuss the characteristics of the traffic generated by our scheme, and how they can be used to be classified as *suspicious* by the warden.

Focusing on the application layer protocol, both communicating parties are only relying on HTTP, and more particularly on the GET request. This request is one of the most common message used on the Internet, as loading a single web page requires several of these requests. Therefore, from a protocol point of view the traffic generated by the communicating parties will look like genuine web traffic to the warden. The warden could use the content (*announce* and *scrape-all* requests) to identify the targegeted application, i.e. BitTorrent. The warden could also come to the same conclusion by analysing the destination of the requests that are BitTorrent trackers (lists of BitTorrent trackers are available on-line[3]). While the application targeted by those HTTP requests, i.e. the tracker component of the BitTorrent system, may not be as common as web services, it is still one of the most popular services of the Internet and has been reportedly used by millions of users[4].

The goal of a warden aware of the protocol could be to detect the communicating parties and the traffic corresponding to our scheme. Also, the warden may already know the identity of one party and could try to identify the other party.

The warden should first focus on the traffic directed to the centralized BitTorrent trackers, and more particularly to trackers providing the functionality required by our scheme (open centralized tracker supporting scrape-all requests). All the BitTorrent clients performing *announce* requests should be considered as potential writers while all the clients performing *scrape-all* requests should be considered as readers. As stated before, the second category can be rather small while the first can be very large. Indeed, the number of clients connected to a tracker at the same time can scale up to 100.000's peers [1]. To the best of our knowledge, no information on the number of clients performing scrape-all requests is available. Nevertheless, given the specific nature of this request, we can conjecture that the number of clients will be small. The latter property makes our system asymmetric in terms of detection difficulty: discovering the identity of the writer is much more difficult than discovering that of the reader. This feature suits well the scenario where an insider is attempting to exfiltrate information to the outside.

Announce requests performed by the writer are using content identifier forged for the purpose

[3] `http://www.torrentking.org/`

[4] `http://torrentfreak.com/bittorrent-surges-to-150-million-monthly-users-120109/.`

of the protocol, and are therefore not corresponding to real content shared on the BitTorrent ecosystem. List of contents shared in the BitTorrent ecosystem can be found on websites like *The Pirate Bay*. The warden could use this information to verify that the identifier corresponds to real content, but in fact there is a significant part of content shared in the BitTorrent ecosystem that is not found on the common *torrent* databases. Therefore, this technique could be used to narrow down the search, but will not be enough to identify the writer.

Finally we want to highlight two features of our scheme that make the detection even more difficult. First, as opposed to many other covert channels, no direct connection is ever established between the communicating parties. This means that the warden may not even suspect that the parties are exchanging information. Second the system is asynchronous, i.e. the read operation can be done several ten's of minutes after the write operation. Thanks to this feature the temporal correlation between the writer and the reader activity can be kept at a low level.

7 Application to Distributed Hash Table

In BitTorrent, the role of the centralized tracker can be replaced by a decentralised alternative, based on distributed hash table (DHT). Applying our communication schemes to the DHT is an appealing solution, as the distributed nature of the DHT makes it highly resilient and difficult to monitor by a warden. However, tracker and DHT differ in a number of aspects that prevent the direct application of our scheme on a DHT. In this section we describe how our covert channel communication scheme can be adapted to a DHT.

7.1 Distributed Hash Tables in BitTorrent

Distributed hash tables, used as decentralized trackers in P2P systems, are used to store a number of *(key,value)* pairs. In a DHT, each node is in charge of storing a subset of the *(key,value)* pairs. This distributed data structure implements a *push* operation to store *(key,value)* pairs and a *get* operation to retrieve a value corresponding to a given key.

A number of P2P file sharing system lookup functionalities are provided by a DHT, with several BitTorrent clients supporting the use of a DHT in order to find peers sharing a specific content. For instance, Vuze, a popular BitTorrent client, uses a variant of the highly efficient Kadelmia DHT [8]. This DHT stores pairs of the form *(info_hash, peer_info)*, where *peer_info* is a tupple *{IP_address, port_number}*. A client joining a swarm contacts the node in charge of the corresponding key (the *info_hash* of the content) to get a list of peers and to add itself to this list. This functionality provided by the DHT is identical to what is provided by a centralized tracker.

However the DHT does not support the *scrape-all* request, that allowed our reader to get the full list of content registered on a centralized tracker. Recall that this feature was essential to our tracker-based system, as the information was hidden in those content identifiers. In fact, the only way to retrieve/store information from/to the DHT is by providing a content identifier.

Since the IP part of the peer_info is set to that of the requesting peer, there is only one piece of data left that can be arbitrarily chosen: the port number. The port number is a 16-bits unsigned integer designating the port on which the BitTorrent peer listens to incoming connections.

7.2 System description

We propose to use the port number to convey information and to use the content identifier as a key for the communication. The atomic write operation is done by adding a new pair *(i, IP :*

port_ number) to the DHT, where i is the content identifier, *IP* is the IP address of the writer and *port_ number* contains the hidden data. In return, the atomic read operation is performed by retrieving the *peer_ info* associated to the content identifier i. The DHT can then be seen as a memory in which the content identifier i is the memory address, while port_number is the data stored at this address. A get request for the content identifier i would normally return only one value IP:port_number, making the extraction of the data straightforward.

Starting from the atomic read and write operations, a more advanced anonymous communication system can be built. The communicating parties will use an ordered list of k content identifier $I = \{D_i\}_{1 \leq i \leq k}$ to designate the address at which the data will be stored in the DHT. In addition some of the features of the original system can be reused. First, the Hash-based filtering can be employed to prevent the genuine DHT activity from interfering with our system. Finally, encryption and erasure coding can be directly applied to provide respectively obfuscation and loss resilience. Note that the indexing feature is not required anymore, as the content identifier list I provides an implicit index.

The management of the content identifier list I could hamper the usability of the system, as the length of the communication must be known in advance and the exchange and storage of those identifiers can be costly. Instead we propose to use a list of identifiers generated by Pseudo Random Number Generator (PRNG) initiated by a seed shared by the communicating parties. Thanks to that, content identifier can be generated on the fly, ensuring a potentially infinite stream of messages while keeping to a minimum preliminary communication as well as memory consumption.

7.3 Performance and detectability

In the Kademlia DHT implemented by Vuze, a piece of data is erased from the node eight hours after the last access [5]. Therefore, once a message was sent to the DHT we can expect that it will remain available to the reader for the same time period (eight hours). We note that the corresponding time window is much larger than for the centralized tracker-based system (where it is at most 3600 seconds) and allows for asynchronous communication in which read and write operations can be performed with a difference of several hours.

DHTs are designed to be highly resilient storage systems. Indeed their distributed nature as well as data replication (each piece of data is replicated over 20 different nodes) make the loss of data an extremely rare event [5].

Using the DHT instead of a centralized tracker would increase the scalability of our system. Indeed, in the centralized tracker based system both read and write operations involve the tracker. The write operation is relatively lightweight, but the read operation puts the tracker under a heavy load as it requires the transfer of a several Megabytes file. Whereas in a DHT-based approach, the random distribution of the data over the DHT nodes distribute the load over a large number of hosts, allowing for the use of a larger number of covert communications in parallel.

The detectability of this scheme, due to the nature of the communication used by the DHT, slightly differs from the centralized tracker-based version. In particular, the read and write operations both use Kadelmia RPC commands embedded in UDP packets. These requests are less common than the HTTP/GET requests used by the centralized tracker-based scheme, and therefore potentially easier to detect. One advantage of the DHT approach though is the large scale system in which the communication can be hidden. Indeed, if it is quite challenging for a warden to monitor the communication over potentially millions of DHT nodes.

8 Related works

Ephpub [2] is a DNS-based ephemeral communication scheme which makes use of DNS server caches to store information. Although the main goal of this system is not to provide anonymous communications, and rather provides ephemeral publication of data (i.e. encrypted data with a lifetime of the key to decrypt it), it shares several aspects with the system proposed in this paper. Indeed, both systems employ a common Internet service to store information, and no direct connection between the communicating parties is required.

Another tool aiming to provide ephemeral publication is called *Vanish* and has been introduced in [5]. In Vanish, the key used to encrypt a message between two communicating parties is stored on the DHT, ensuring that the message becomes unreadable once the key disappears because of information erasure (either due to expiration delay or churn) on the DHT. As in the DHT extension described in our work (Section 7) the sender uses the DHT to store pieces of information that will be later retrieved by a receiver informed of their locations. However, Vanish leverages the DHT to only store bits of the crypto-system keys. Vanish makes use of the Shamir secret sharing scheme, and randomly stores bits of the key on different nodes of the DHT (used as an index storage). Our proposed DHT-based asynchronous covert communication scheme exploits the DHT to store the data itself and relies on the tracking capabilities of the DHT, rather than the storage property.

In [13], authors present *Overbot*, a stealth command and control channel using the Kadelmia distributed hash table. The objective of this system is to allow compromised nodes to declare themselves to the botmaster and then to receive commands. The botmaster deploys a number of *sensor nodes* in the DHT, that are in charge of receiving command requests from compromised nodes. Those requests are embedded into genuine DHT requests but include an encoded message that can only be recognized by the botmaster. Once a request for command is received, the bostmaster redirects the compromised node to another controlled node of the DHT, an *actuator node*, that is in charge of delivering the command. Similarly to our system, Overbot hides its communication within legitimate traffic so that those communications appear to be genuine traffic for an external observer. However, in *Overbot* a direct connection is established between the different entities (sensor/actuator nodes and compromised nodes), while in our scheme, direct communications between the parties never occurs.

Use of the BitTorrent system to provide a covert channel has been previously investigated in [4], where authors propose implementation of covert channels using various functionality of the BitTorrent system. One of the covert channels exploits the fact that the order in which the data blocks are exchanged is left unspecified (on purpose). The conspiring nodes can then perform a mutual identification by exchanging a shared secret encoded by the requested block order. As opposed to our system, this work only considers communication through direct exchange of data between the communicating parties. This could be successfully exploited in a de-anonymization process if one of the parties involved into the communication is identified. communication through direct exchange of data

A number of P2P systems providing anonymity have been proposed, like GNUnet and Freenet[5]. We note that the purpose of these systems is to provide anonymous communications and not covert communications like our proposal. Indeed, the identity of the communicating parties is hidden, but the data exchange itself is not hidden at all.

[5]Freenet `https://freenetproject.org/` and GNUnet `https://gnunet.org/`

9 Conclusion

We have presented a new covert channel based on a widely used service available on the Internet: centralized BitTorrent trackers. Our system does not require the cooperation of the tracker and allows the sending of data at a moderate rate. We have implemented a prototype that demonstrates the practicality of our proposal and evaluated the performance of our system. By analyzing the scheme under two attacker models, with the monitoring warden being either the strong or the weak adversary, we have shown that the nature of the communications involved in the proposed scheme makes the detection and blocking of the communications difficult. Finally, we have presented an extension of this scheme that relies on the decentralized, DHT based, trackers in the BitTorrent system.

References

[1] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on cooperation in bittorrent communities. In *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, P2PECON '05, pages 111–115, New York, NY, USA, 2005. ACM.

[2] C. Castelluccia, E. De Cristofaro, A. Francillon, and M. A. Kaafar. Ephpub: Toward robust ephemeral publishing. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, 2011.

[3] Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*, Dec. 1985. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[4] R. Eidenbenz, T. Locher, and R. Wattenhofer. Hidden communication in P2P networks steganographic handshake and broadcast. In *INFOCOM, 2011 Proceedings IEEE*, pages 954 –962, april 2011.

[5] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proc. of the 18th USENIX Security Symposium*, 2009.

[6] N. Lucena, G. Lewandowski, and S. Chapin. Covert channels in ipv6. In *Privacy Enhancing Technologies*, volume 3856 of *Lecture Notes in Computer Science*, pages 147–166. Springer Berlin / Heidelberg, 2006.

[7] X. Luo, E. Chan, and R. Chang. Crafting web counters into covert channels. In *New Approaches for Security, Privacy and Trust in Complex Environments*, volume 232 of *IFIP*, pages 337–348. Springer Boston, 2007.

[8] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[9] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Technical Report CS-96-332, University of Tennessee, July 1996.

[10] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

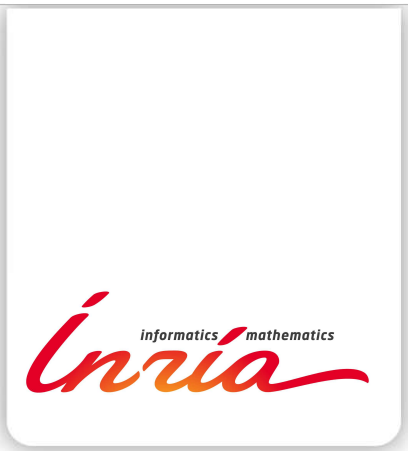
[11] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2), Apr. 1997.

[12] G. J. Simmons. The prisoners' problem and the subliminal channel. In D. Chaum, editor, *CRYPTO*, pages 51–67. Plenum Press, New York, 1983.

[13] G. Starnberger, C. Kruegel, and E. Kirda. Overbot - a botnet protocol based on kademlia. In *4th International Conference on Security and Privacy in Communication Networks*, 2008.

[14] M. Wolf. Covert channels in lan protocols. In *Local Area Network Security*, volume 396, pages 89–101. Springer Berlin / Heidelberg, 1989.

Contents



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399