Peer-to-Peer Traffic Localization as a Service

Moritz Steiner and Matteo Varvello
Bell Labs, Alcatel-Lucent, USA
{*first.last*}@alcatel-lucent.com

***Abstract*—Peer-to-Peer (P2P) file-sharing applications generate extremely high network transit costs for the ISPs due to their network agnostic nature. A number of strategies have been deployed to localize P2P traffic, mostly relying on the communication between the peers and the directory servers (trackers). However, recently many users favor distributed tracking based upon Distributed Hash Tables (DHTs). In this demonstration, we present the first traffic localization mechanism for DHT-based file-sharing networks. It runs as a service in the cloud. Thus, it does not require any equipment installation at an ISP, nor cooperation between an ISP and P2P networks.**

I. Introduction

BitTorrent is by far the most popular Peer-to-Peer (P2P) protocol, adopted by several file-sharing applications such as μTorrent [10] and Azureus [3]. The BitTorrent protocol ignores the location of peers in the network when determining how files are exchanged, thus generating a large volume of expensive inter-ISP traffic. In the past, tracker-based traffic localization strategies have shown they effectively reduce BitTorrent inter-ISPs traffic [13].

Recently, BitTorrent introduced a distributed tracking feature: peers can now locate which peers hold a copy or a portion of a file by querying a Distributed Hash Table (DHT). This feature, originally thought as a backup solution for the trackers, is now supported by most BitTorrent client implementations and it attracts a large share of BitTorrent user requests [11]. The addition of a DHT to the BitTorrent network decreases from the effectiveness of tracker-based traffic localization mechanisms.

In [11], we designed and prototyped the first traffic localization mechanism for DHT-based file-sharing networks. Our prototype targets the BitTorrent Mainline DHT [6] and can be run as a service in the cloud. No equipment installation is required at an ISP, nor does it require cooperation between an ISP and P2P networks (which is unlikely to occur). This demonstration shows the prototype used in the experimental evaluation of [11].

II. Design Overview

Our localization mechanism works in two steps. First, we intercept all the messages from peers announcing in the DHT that they hold a file or a portion of it. Then, we intercept all the requests for this file and answer with *local* peer-sets, i.e., sets of peers located at the same ISPs as the requesting peers. We now describe both steps in detail.

A single entity can join a P2P network many times with many distinct logical identities called *sybils* [4]. To intercept announces and requests for a file, we insert several sybils in the DHT with nodeIDs close to the info_hash of the file [9]. The sybils are constantly aware of the peers that hold the file as well as the peers requesting it. Under this premise, localization is straightforward. Our sybils simply respond to the queries for this file with localized peer-sets. If only a few local peers are available, external peers are used to complete the peer-set.

Our localization mechanism targets only popular files for two reasons. First, only the traffic associated with files requested by more than one peer from the same ISP at the same time has potential for localization. Second, we aim to minimize the number of files to be localized. In fact, the number of sybils we need to insert in the DHT to achieve traffic localization scales linearly with the number of files that we localize.

III. Mainline Implementation

As a proof of concept, we implemented a prototype of our localization mechanism for the Mainline DHT. We pick the Mainline DHT since it has the largest user base with more than eight million active users at any point in time. Note that our localization mechanism can easily be ported to any other DHT-based P2P network build upon the Kademlia protocol [8], e.g., Azureus [3], Emule [5], and Ares [2].

We now briefly overview the relevant messages used in the Mainline DHT. For more details, the interested reader is referred to [6].

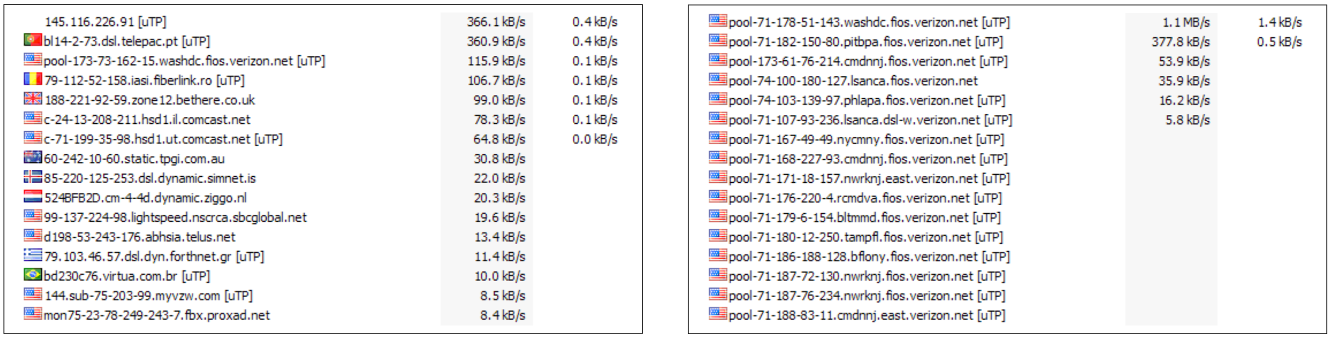
`get_peers(`I`)` – it performs a lookup operation, i.e., retrieve the list of peers holding a copy or a portion of a file with info_hash I. The lookup works iteratively. At each intermediary hop of the iteration, peers respond to a `get_peers` message with the IP addresses, ports, and nodeIDs of the peers closest to I. At the final hop of the iteration, peers return the IP addresses and ports of the peers that hold a copy or a portion of the file with info_hash I.

`announce_peer(`I`)` – it is used by a peer P to announce that it hosts a file with info_hash I. P sends several `announce_peer` messages to the peers in the DHT whose identifiers are the closest to I (previously retrieved using a `get_peers` message).

`ping(`P`)` – it checks that peer P is alive.

We now focus on the localization of the traffic associated with a single file identified by info_hash I.

In the first step of the localization mechanism, we intercept the `announce_peer` messages for I. To do so, we insert sybils in the DHT with nodeIDs closer to I than any real peer. The insertion of the sybils in the DHT consists of informing real peers whose nodeIDs are close to I about the existence



(a) Localization turned off.

(b) Localization turned on.

Fig. 1. Screenshots of the peers a μTorrent client downloads from.

of the sybils. These peers will then propagate this information to other peers in the DHT. We proceed as follows:

- First, we discover the peers whose nodeIDs fall within Z, the portion of the hash-space that shares a prefix of at least z bits with I. To do so, we send out multiple `get_peer` messages with target info_hashes close to I.
- Then, we send `ping` messages from the sybils to all peers in Z. This operation populates the routing tables of the nodes in Z with information about our sybils.

The information derived from the received `announce_peer` messages is stored in a database common to all the sybils. We use Maxmind [7] to resolve a peer's IP address to its ISP. For an entry in the database, we use the same timeout policy as currently implemented by the BitTorrent clients.

In the second step of the localization, we intercept the `get_peers` messages for I and we reply to them with local peer-sets. Similarly to the `announce_peer` messages, the sybils also intercept the `get_peers` messages during their iteration across the DHT. We construct the replies to the `get_peers` messages as follows. First, we determine the ISP of the querying peer using Maxmind. Then, we form the peer-set to be returned by the sybil searching in the shared database for peers located at the same ISP as the requesting peer. In case not enough local peers are found, we complete a peer-set with external peers.

To localize more than one file, we repeat the outlined procedure. Resource consumption scales linearly with the number of files to localize.

IV. Demonstration Setup

The demonstration runs on the live Mainline DHT network and involves: a *server side*, that shows the sybils running on an Amazon EC2 [1] machine, and a *client side*, that shows several clients located at different ISPs for which we enable the localization. In the remainder of this section, we describe both the server and client side in detail.

Server side – We strategically place the sybils in the DHT in order to localize the traffic associated with several popular files. In order to demonstrate that no physical installation is required at an ISP, we run the sybils on an Amazon EC2 machine. For each localized file we show popularity statistics, i.e., the number of peers per ISP holding the file or a portion of it. Performance statistics, such as incoming and outgoing messages per second, memory and cpu utilization, and Amazon EC2 related costs, are visualized as well.

Client side – We launch several μTorrent clients located at multiple ISPs and we start the download of the files localized by the sybils. For each peer the client contacts to retrieve a file, we show its geographic location along with the connection speed and the amount of traffic exchanged. Figure 1 shows screenshots of a μTorrent client running in the Verizon ISP while downloading a file with and without localization. For each line, the first column shows the country and ISP of a source, the second column shows the download speed, and the third column shows the upload speed. The goal of this visual support is to demonstrate that the BitTorrent traffic effectively remains within an ISP. For example, the screenshot in Figure 1(a) shows that most of the traffic is non-local when the localization is disabled, whereas Figure 1(b) shows that 100% of traffic is kept within the ISP of the downloading peer when the localization is enabled. Note the high download speed that can be reached connecting to local nodes in the latter case. During a download, we also visualize the DHT control traffic using Wireshark [12] in order to highlight the messages exchanged between a client and our sybils. Specifically, we aim to show that all `announce_peer` messages are correctly received by the sybils.

References

[1] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.
[2] Ares. www.ares.net/.
[3] Azureus/Vuze. http://www.azureus.sourceforge.net/.
[4] J. R. Douceur. The Sybil Attack. In *IPTPS*, Cambridge, MA, USA, March 2002.
[5] eMule. www.emule-project.net/.
[6] Mainline DHT Specification. http://www.bittorrent.org/beps/bep_0005.html.
[7] Maxmind. http://www.maxmind.com/.
[8] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *IPTPS*, Cambridge, MA, USA, March 2002.
[9] M. Steiner, E. W. Biersack, and T. En-Najjary. Exploiting KAD: Possible Uses and Misuses. *Computer Communication Review*, 37(5), Oct. 2007.
[10] uTorrent. http://www.utorrent.com/.
[11] M. Varvello and M. Steiner. Traffic Localization for DHT-based BitTorrent Networks. In *Networking*, Valencia, Spain, May 2011.
[12] Wireshark network protocol analyzer. http://www.wireshark.org/.
[13] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz. P4P: provider portal for applications. In *SIGCOMM*, Seattle, WA, USA, Aug. 2008.