Institut Eurécom
Networking and Security Department
2229, route des Crètes
B.P. 193
06904 Sophia-Antipolis
FRANCE

Research Report RR-08-223

Crawling AZUREUS

June 4th, 2008

Moritz Steiner and Ernst W. Biersack

Tel : (+33) 4 93 00 81 00
Fax : (+33) 4 93 00 82 00
Email : {steiner,erbi}@eurecom.fr

[1]Institut Eurécom's research is partially supported by its industrial members: BMW Group Research & Technology - BMW Group Company, Bouygues Télécom, Cisco Systems, France Télécom, Hitachi, SFR, Sharp, ST Microelectronics, Swisscom, Thales.

1 Introduction

In this paper we present the results we obtained by crawling the DHT of the BitTorrent client AZUREUS for several months. Before this study we crawled the DHT KAD that is used by the eMule and aMule clients and that runs for the moment in parallel to the server based eDonkey network. In KAD or eDonkey all peers are connected together in one big peer-to-peer network, not as it is the case in BitTorrent in a big number of independent swarms. We used the same crawling technique to crawl the DHT Overnet as well as the network of the Storm worm.

Crawling the DHT implemented by a BitTorrent client is a new approach to learn about the behavior of BitTorrent peers, since up today the classical approach was to learn about torrents on portal websites and than to contact the corresponding trackers. Since in BitTorrent every torrent forms a swarm of peers, all existing torrents need to be tracked in order to get a full view of all clients participating. Our approach is not based on torrents but we make use of the DHT in which all peers participate and in which information about all torrents is published. For the moment this DHT is uses as a fall back mechanism in case the tracker goes down. However we have to admit that also with our method of crawling a DHT we do not learn about all peers using BitTorrent but only of those that a client that is compatible to the DHT of AZUREUS . There are several incompatible implementations of DHT by different BitTorrent client. However keeping track of a handful DHTs is more feasible than keep track of hundred of thousands of torrents, knowing that only those published on websites can be tracked.

Crawling these different networks we can make some general remarks. Each network is predominant in one region of the world, KAD in Europe and China, Azureus in North America. The median session times of the peer is in the order of hours and most users are on-line in the evening of their local time zone. A big fraction of the users are connected to the Internet with a classical ADSL line, the ten largest ISPs for the users of both peer-to-peer network are only ADSL providers. The major difference in the implementation is that in KAD the DHT identifiers are chosen randomly once and than permanently stored whereas in AZUREUS the ID depends on the IP and port of the client. Since most users get their IP address dynamically reassigned by their providers every 24 hours, also their ID changes frequently. Therefore it is not possible to track peers in AZUREUS over several sessions.

2 Background on Azureus

The DHT Kademlia [7] is implemented by several peer-to-peer applications such as AZUREUS [2], Overnet [8], eMule [4], aMule [1], and lately the Storm worm [5]. The two open–source projects eMule and aMule share the same implementation of Kademlia KAD and do have the largest number of simultaneously connected users, 3 – 4.5 million users [14], followed by AZUREUS with about 1 million users.

Similar to other DHTs like Chord [15], Can [9], or Pastry [10], Each AZUREUS node has a global identifier. First a string is built up out of the IP and the port number modulo divided by 1999, which is a prime, divided by ":". The SHA1 hash of this string is the DHT ID of the node. Thus the number of AZUREUS nodes per IP address is limited to 2000, since a client running on port 1999 has the same ID than a client running on port 3998 on the same IP address. Moreover since most peers do have dynamically assigned IP addresses by their provider every time they reconnect to AZUREUS they change their identifier in AZUREUS.

3 Measurement Methodology

3.1 Crawlers

The first step in order to do any latency measurements is to learn about the hosts you want to measure to. Therefore we used our crawler Blizzard for KAD [14] and adapted it to work for AZUREUS as well. Our crawler logs for each peer P

- the time of the crawl
- the IP address of P
- the port number used for the control traffic
- the Vivaldi network coordinates (version 1 with 2 dimensions, the height vector, and the estimated error, and version 2 with 4 dimensions, the height vector, the estimated error, and the age of the coordinate) [3, 6]
- the estimated number of peers in the network. The number of participants is estimated by computing the peer density in a small fraction of the DHT and interpolating this number.

The implementation of Blizzard is straightforward: It runs on a local machine, and starts by contacting a seed, which is ran by us. The crawler asks the seed peer for a set of peers to start with and uses a simple breadth first search and iterative queries. It queries the peers it already knows to discover new peers.

First, the number of new peers discovered grows exponentially before it approaches asymptotically the total number of peers in the system. The crawl is done when no new peers are discovered and all the discovered peers have been contacted.

The crawler implements two asynchronous threads: one thread sends the `REQUEST_FIND_NODE(id)` messages (Algorithm 1) and the other thread receives and parses the `REPLY_FIND_NODE` messages returned (Algorithm 2). A list that contains all the peers discovered so far is used and maintained by both threads: the receiving thread adds the peers extracted from the `REPLY_FIND_NODE` messages to the list, whereas the sending thread iterates over the list and sends n (8 to 16) `REQUEST_FIND_NODE(id)` messages to every peer in the list. The value of the DHT ID `id` is different in each of the `REQUEST_FIND_NODE(id)` messages. This is done in order to minimize the overlap between the sets of peers returned.

Algorithm 1: send thread (is executed once per crawl)

```
Data: peer: struct{IP address, port number, DHT ID}
Data: shared list Peers = list of peer elements
/* the list of peers filled by the receive thread and
   worked on by the send thread                          */
Data: int position = 0
/* the position in the list up to which the peers have
   already been queried                                  */
Data: list ids = list of 16 properly chosen DHT ID elements
1 Peers.add(seed); /* initialize the list with the seed peer  */
2 while position < size(Peers) do
3     for i=1 to 16 do
4         dest = Peers[position].DHT ID ⊕ ids[i]; /* normalize bucket to
              peer's position                              */
5         send REQUEST_FIND_NODE(dest) to Peers[position];
6     position++;
```

3.2 The statistic message

Beside the `REQUEST_FIND_NODE` messages we send a second type of query to every AZUREUS peer: the `REQUEST_STATS`. The answer to this request is the `REPLY_STATS`

Algorithm 2: receive thread (waits for the `REPLY_FIND_NODE` messages)

```
   Data: message mess = REPLY_FIND_NODE message
   Data: peer: struct{IP address, port number, DHT ID}
   Data: shared list Peers = list of peer elements
   /* the list shared with the send thread                    */
1  while true do
2      wait for (mess = REPLY_FIND_NODE) message; foreach peer ∈ mess do
3          if peer ∉ Peers then
4              Peers.add(peer);
```

messages that contains a large number of useful information about the peer:

- the detailed version number, e.g. 3.0.5.2.,
- the DHT protocol version,
- the uptime,
- the cumulated bytes/packets sent and received for maintenance overhead,
- the average bytes/packets sent and received for maintenance overhead,
- the number of pings received,
- the number of different type of requests received,
- number of contacts, number of nodes and leaves (buckets) of the routing tree,
- and the number of keys stored in the database.

4 Results

4.1 Uptimes

Every peer counts the time elapsed since it has been launched. The median uptime is of 8 hours, 5% of the peers have been up longer than 6 days and one 1% longer than 26 days (Figure 1). This is very long compared to a median of 2 1/2 hours for the session times in eMule and aMule [11].

The distribution of the uptimes is heavy tailed, since it decays slower than exponentially.

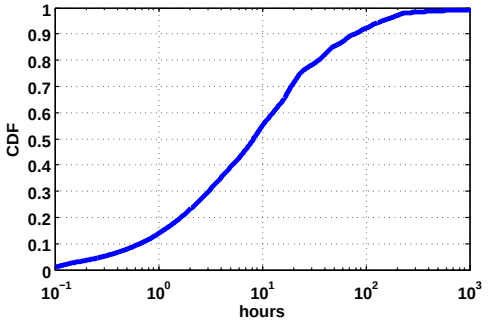


Figure 1: CDF of the uptimes of the AZUREUS clients.

4.2 Versions

In total we saw 191 different versions of clients during our crawls. In Figure 2 the 10 most used version on April 25th 2008 1300 CET are plotted. Note that version 3.0.5.2 is the most up to date version. In fact the latest versions of the AZUREUS client automatically updates itself, that is why more than 75% of the peers use that version. The automatic update is able to reach in only some days most of the users running a version including the automatic update. In Figure 3 one example of a version change is plotted. On March 5th 2008 the new version 3.0.5.0 is introduced and replaces in some days the version 3.0.4.2. You may notice that until April 25th again an new version, 3.0.5.2, has been introduced.

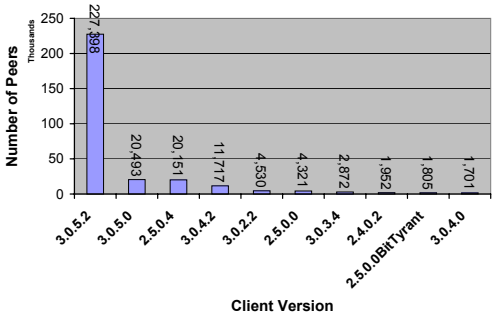


Figure 2: Most used versions on April 25th 2008.

The DHT protocol itself has also different version. 97.2% of the clients use the latest version 16 (VIVALDI_FINDVALUE), 2.2% use the version 15 (GENERIC_NETPOS) and 0.6% use the version 14 (VENDOR_ID and BLOCK_KEYS). Version 14 does not transmit any Vivaldi coordinates, version 15 transmits the old version V2.2 of the Vi-

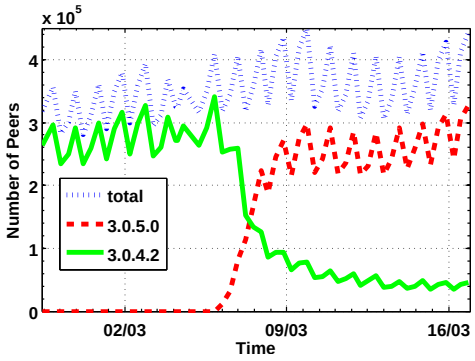


Figure 3: Quick Rollout of a new version.

valdi 2 coordinates (the most recent version is V2.4), this version uses 5 dimensions and not 4 dimensions plus a height vector.

4.3 Geographical Origin

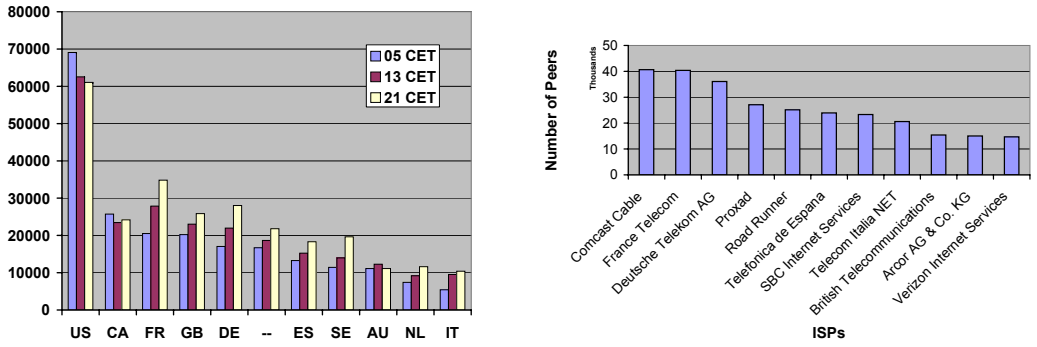
Peers from the US dominate the AZUREUS network. They are followed by peers from Canada and western Europe. In total peers from over 190 different countries have been discovered. Figure 4(a) shows the distribution of the most important countries on April 25th 2008 at different times of the day. Note that most users use AZUREUS in the evening hours.

A histogram of the most used Internet providers can be found in figure 4(b). According to their countries of origin the biggest providers are found in the US, France, Germany, Spain, and Italy.

The users behavior seems to be similar to that one observed in KAD. However in KAD Chinese and European peers dominate, peers from the US play a negligible role only [11].

4.4 IP and port

To get an idea where from in the Internet the AZUREUS users come from we plot the IP addresses on the *Map of the Internet* from the xkcd comic (Figure 5). The comic orders IP addresses on a Hilbert curve, and marks the /8 blocks by their original allocation. Any consecutive string of IP addresses will translate to a single compact, contiguous region on the map. Each of the 256 numbered blocks represents one /8 subnet.

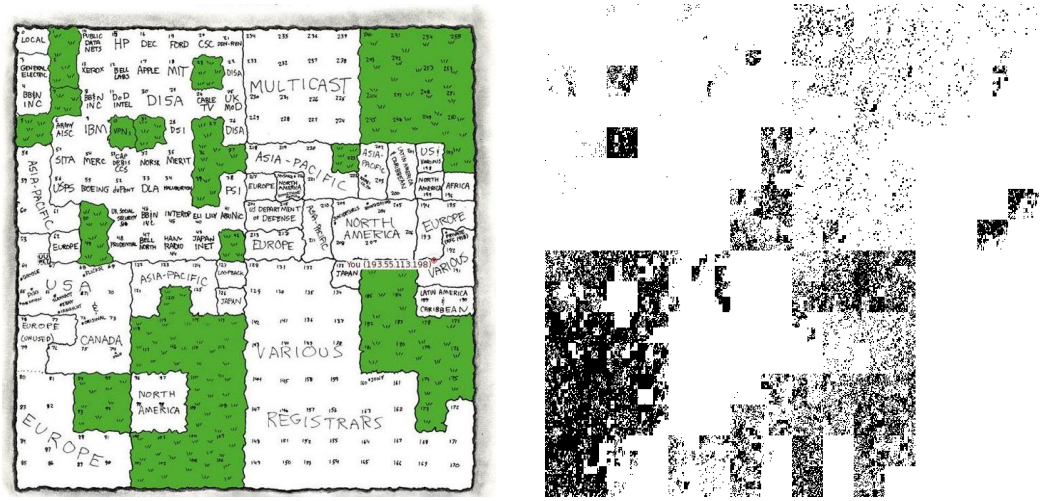


(a) Geographical origin of the AZUREUS peers.

(b) Internet provider of the peers.

Figure 4: Country and ISP of origin.

Most of the users come from IP space allocated to the US, Canada, Europe, and various registrars. However there are some users from class A networks belonging to BBN, MERC, and others.



(a) Address space

(b) IP origin of AZUREUS peers

Figure 5: Map of the Internet

Only bout 5% of the peers make use of the default BitTorrent port 6881. The other peers use randomly assigned port numbers above 50,000 (Figure 6).

4.5 Overhead

The statistics also include information about the overhead needed to maintain the DHT. The median amount of data a peer receives is 286 bytes (2 packets) per second and

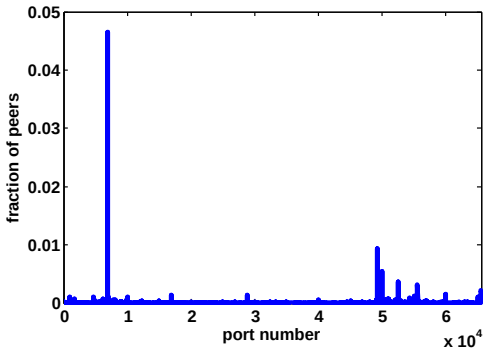
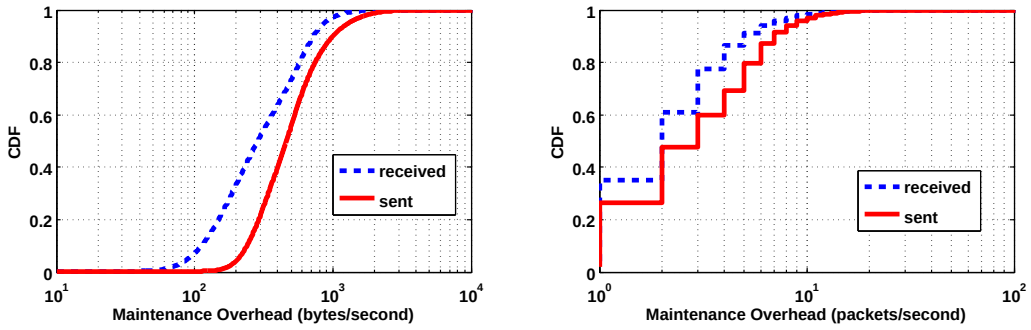


Figure 6: Histogram of the ports used by the AZUREUS peers.

sends is 462 bytes (3 packets) per second (Figure 7). For comparison the data of a KAD client. We dumped all the communication needed for the maintenance of the DHT over 24 hours. It summed up to 4 Packets per second or 422 bytes per second, which is less compared to AZUREUS. AZUREUS also keeps track of the cumulated maintenance overhead since the client has been started. The median of the peers received 10.65 megabytes (81,907 packets) and sent 14.5 megabytes (119,005 packets) (Figure 8).

The difference between the amount of overhead sent and received is due to the fact, that some messages are sent to stale peers. These messages are accounted for on the sending side, but they are never received by anyone. From the collected data we can deduce that about one third of the contacted peers are stale. This is about the same number we obtained for KAD [13].

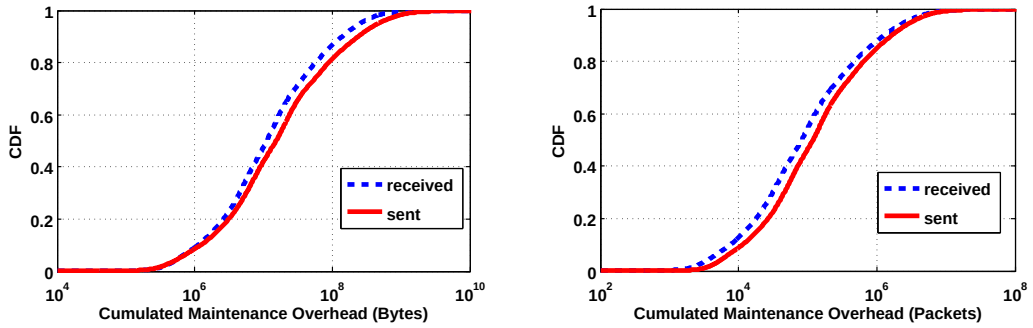


(a) Bytes per seconds

(b) Packets per second

Figure 7: Maintenance overhead

Having a look at the different message types on can notice a huge difference to the values we found in KAD[12]. In AZUREUS the median number of store requests per second is 0.07, the median number of find value request is 0.31 (Figure 9). In



(a) Bytes per seconds (b) Packets per second

Figure 8: Cumulated maintenance overhead

AZUREUS there are 4 times more find value requests than store requests, in KAD this is the opposite, there are 10 times more publish requests than search requests. This is due to two main facts: First in KAD keyword and sources are published, whereas AZUREUS has only one layer of metadata. Given that per source there are 5 keywords in average the amount of message is already multiplied by 6. Second the republishing interval of a content in AZUREUS is of 8 hours, whereas sources in KAD are republished every 5 hours.

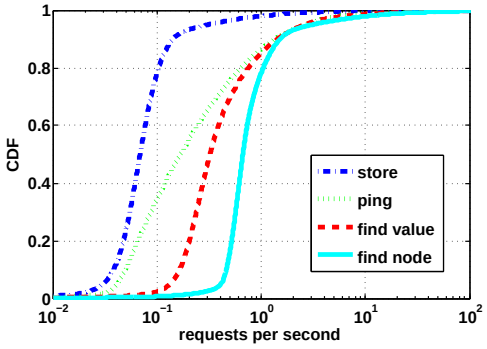


Figure 9: CDF of the different requests received per second per node.

The median number of entries of the DHT a node is stores in its local hash table is 402 (Figure 10). Long living nodes store up to ten thousands values, the maximum we found was 34,807.

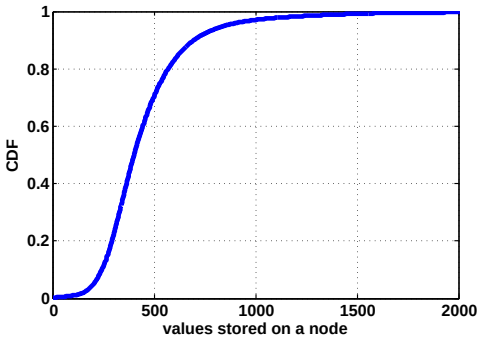


Figure 10: CDF of the number of entries stored per node.

4.6 Estimated peer number

For one crawl we evaluated the accuracy of the estimation of the peers in the network. Our crawler found 1,040,943 peers, the average estimation is 1,101,500, the median is 1,097,223. The minimum is 2, this is most probably a peer still in the bootstrap phase, the maximum is 12,461,686. This estimation is very accurate, the deviation compared to the number of peers we found by crawling is less than 5%.

The estimation of the size of the DHT implemented in AZUREUS is based on a density function. N_0 is the node himself, N_1 the nearest peer, N_2 the 2nd nearest peer $\ldots N_p$ the p_{th} nearest peer that in the routing table of N_0. p is set to 20, since there are at most 20 peers in one bucket of the routing tree, in this case it is the deepest bucket containing the closest contacts. Let D_i be the XOR distance between N_0 and N_i. The local estimation of the DHT size is done in the following way:

$$n = \frac{2^{160}}{\frac{\sum_{i=1}^{p} i\, D_i}{\sum_{i=1}^{p} i^2}}$$

The enumerator is the maximum possible number of nodes in the DHT, the denominator is the filling level of the hash space. It is 1 if every position is filled, if every hash value is taken. Consider the case of $p = 2$, the first node has a distance of 2 to us, there is a space for only one other node in between, the second node has a distance of 4, there is one possible space between the 2nd node and the 1st node. In other words every second possible place is taken. In this example the denominator would be $\frac{1*2+2*4}{1+4} = 2$; the estimation for the number of nodes is going to be 2^{80} since every second place in the hash space is empty.

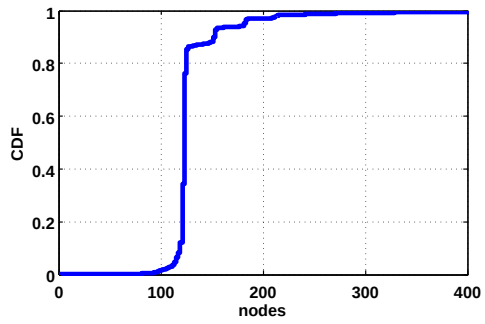
In a second step this value is averaged with the estimation learned from the other

nodes except for the three smallest and the three biggest values.

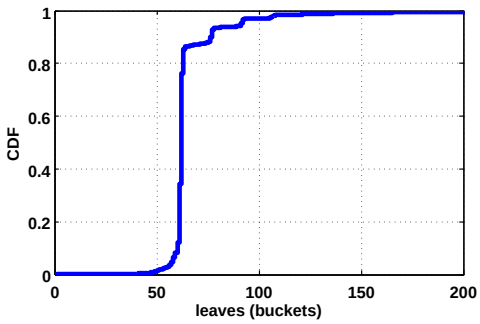
4.7 Contacts

The contacts in AZUREUS are organized in a tree. The median of the number of nodes in the tree is 123 (75% of the peers have values between 119 and 125), for the number of leaves (buckets) attached to the tree it is 63 (75% of the peers have values between 60 and 63). The median of the number of contacts in the routing tree is 1,107 (80% of the peers have values between 1,050 and 1,155) (Figure 11). This is about the square root of the number of participants $O(\sqrt{n})$, much more than the proposed DHT routing table size in literature that grows logarithmically with the number of participants $O(logn)$. Since the KAD network does have the same properties and in KAD the average number of hops needed to find a content is 2.5, we believe that this is also true for AZUREUS.

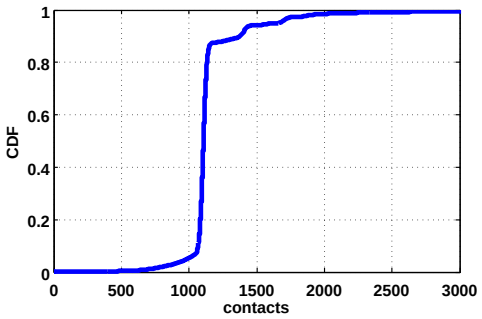
Calot [16] ensures 2 hop routing to any destination with a routing table of $O(\sqrt{n})$ contacts, this is slightly better than the number of hops needed by AZUREUS.



(a) CDF of the number of nodes in the routing tree.



(b) CDF of the number of leaves (buckets) in the routing tree.



(c) CDF of the number of contacts in the routing tree.

Figure 11: Data about the routing tree.

References

[1] A-Mule. `http://www.amule.org/`.

[2] Azureus. `http://azureus.sourceforge.net/`.

[3] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proceedings ACM SIGCOMM 2004*, Aug. 2004.

[4] E-Mule. `http://www.emule-project.net/`.

[5] T. Holz, M. Steiner, F. Dahl, E. W. Biersack, and F. Freiling. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In *First Usenix Workshop on Large-scale Exploits and Emergent Threats (LEET)*, San Francisco, CA, USA, Apr. 2008.

[6] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. In *4th USENIX Symposium on Networked Systems Design & Implementation*, pages 299–311, 2007.

[7] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer informatiion system based on the XOR metric. In *Proceedings of the 1^{st} International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 53–65, Mar. 2002.

[8] Overnet. `http://www.overnet.org/`.

[9] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, 2001.

[10] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale Peer-to-peer systems. In *Proceedings of Middleware*, Heidelberg, Germany, November 2001.

[11] M. Steiner, E. W. Biersack, and T. En-Najjary. Actively Monitoring Peers in Kad. In *Proceedings of the 6^{th} International Workshop on Peer-to-Peer Systems (IPTPS'07)*, 2007.

[12] M. Steiner, E. W. Biersack, and T. En-Najjary. Exploiting KAD: Possible Uses and Misuses. *Computer Communication Review*, 37(5), Oct 2007.

[13] M. Steiner, D. Carra, and E. W. Biersack. Faster content access in kad. In *Submitted to Proceedings of the 8th IEEE Conference on Perr-to-Peer Computing (P2P'08)*, Aachen, Germany, Sept. 2008.

[14] M. Steiner, T. En-Najjary, and E. W. Biersack. A Global View of KAD. In *Proceedings of the Internet Measurement Conference (IMC)*, 2007.

[15] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-to-peer lookup service for Internet applications. In *Proceedings of SIGCOMM*, pages 149–160, San Diego, CA, USA, 2001. ACM Press.

[16] C. Tang, M. J. Buco, R. N. Chang, S. Dwarkadas, L. Z. Luan, E. So, and C. Ward. Low traffic overlay networks with large routing tables. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 14–25, New York, NY, USA, 2005.