Measurement of *eDonkey* Activity with Distributed Honeypots

Oussama Allali, Matthieu Latapy and Clémence Magnien

LIP6 – CNRS and Université Pierre et Marie Curie (UPMC – Paris 6)
104, avenue du Président Kennedy
75016 Paris – France
`firstname.lastname@lip6.fr`

***Abstract*—Collecting information about user activity in peer-to-peer systems is a key but challenging task. We describe here a distributed platform for doing so on the *eDonkey* network, relying on a group of *honeypot* peers which claim to have certain files and log queries they receive for these files. We then conduct some measurements with typical scenarios and use the obtained data to analyze the impact of key parameters like measurement duration, number of honeypots involved, and number of advertised files. This illustrates both the possible uses of our measurement system, and the kind of data one may collect using it.**

Index Terms*—peer-to-peer, measurement, honeypot, *eDonkey

I. Introduction.

Peer-to-peer now accounts for the use of 70% or more of the global internet bandwidth [1], [2]. It is therefore important to understand the underlying mechanisms, in order to design efficient protocols. In particular, user behaviors and data properties are of high interest: are there correlations among the files owned by each user? are there communities of interest? are there sets of similar data or very popular data? how does data popularity evolve? ...

Much work has been done to answer these questions [3], [4], [5]. The most widely used approach is to watch the activity in a network of interest for a period of time. This raises important difficulties due mainly to the distributed and anonymous nature of peer-to-peer systems, their dynamics, and their sheer size.

This paper explores a new method for collecting information on what occurs in one of the main peer-to-peer systems currently in use, *eDonkey*[1]. It consists in introducing *honeypots* in the network, *i.e* peers pretending to offer files and logging all the queries they receive for these files from other peers.

We begin by presenting the current state of the art of peer-to-peer measurement (Section II), and then we describe the new solution we propose (Section III). Finally we will illustrate its relevance with some practical measurements (Section IV), before presenting our conclusions (Section V).

II. Related work.

Measurement of peer-to-peer systems is a very active area of research. We focus here on the measurement of peer activity, thus ignoring measurements of peer-to-peer overlays and protocols, which are out of the scope of this paper. Several approaches have been used to collect information on peer-to-peer activity, each with its own advantages and drawbacks. We rapidly describe them in this section.

A. Measurement at server level.

In centralized or semi-centralized systems (like *eDonkey*), queries sent by peers are managed by servers; it is then possible to passively collect data directly on these servers. The measurement setting then consists in either modifying the server so it may log the queries it manages [7], [8], or capturing traffic at IP level and then decoding it [9].

This method has the advantage of collecting *all* the information managed by a given server. However, as actual file exchanges occur between peers (out of the sight of servers), this information is not captured. Moreover, it requires cooperation with server administrators.

B. Measurement at peer level.

Passive measurements are also possible in fully distributed systems at client level: a modified client may observe the traffic going through it including in some cases keyword queries, file searches, etc.

In [3], [4], [10], [11] authors set up such measurements. The main issue of this approach is the need for users that agree to cooperate, which limits the amount of data obtained. To increase it, the author of [12] designed a large distributed architecture called GnuDC (Gnutella Distributed Crawler); it monitors the network by being attached to a large number of peers.

C. Measurement by client sending queries.

An active measurement method from clients is also possible. It consists in designing a client that sends queries in the system and records the obtained answers (lists of files and providers, typically). This has been done in *Napster* [13] and *eDonkey* [14], [15] with success. The main drawback of this approach is that it is active: it may interfere with the observations, and the rate at which queries may be sent is limited.

[1] *eDonkey* is a semi-distributed peer-to-peer file exchange system based on directory servers. An unofficial documentation of the protocol is available [6].

D. Measurement at ISP level.

Finally, one may capture peer-to-peer traffic directly on ISP infrastructures, in a passive way. In [5], [16], [17] for instance, data is collected from several routers, and different peer-to-peer applications (Gnutella, FastTrack, DirectConnect, *eDonkey*) are observed.

This approach provides network-level data, with limited information on users and exchanged files. This makes it quite different from other approaches discussed here. Moreover, it relies on cooperating with ISPs, which have limited rights to observe user traffic.

III. Our data collection system.

Our measurement infrastructure consists in a set of fake peers (the honeypots) connected to different servers, and a manager controlling these honeypots. We first present our manager and honeypots, and then we discuss privacy concerns.

A. Manager.

The manager's role is to set up the honeypots, and then coordinate them and centralize the data they collect.

The first function of the manager is to launch the honeypots. It specifies to each of them a server to connect to. Each honeypot then attempts to connect to the server, and reports its status (connected or not), as well as its *clientID*[2], if relevant, to the manager. This makes it possible to re-launch dead honeypots or to redirect them toward other servers. The manager regularly checks the status of each honeypot for the same reason.

Several strategies make sense for assigning honeypots to servers. One may typically choose a different sever for each honeypot, in order to obtain a more global view. The choice of servers may also be guided by their resources and number of users, so that the honeypots may reach the largest possible number of peers.

The second function of the manager is to tell honeypots to advertise fake files. It specifies the name, size and *fileID*[3] of each file. Again, many strategies are possible to choose the files to advertise, and the manager is in charge of implementing the chosen strategy. For instance, it is possible to study the activity on a specific topic by choosing the files accordingly. One may then ask to the discovered peers their list of shared files and add these files to the the honeypot's list. One may also advertise random files.

Finally, the manager periodically gathers the data collected by honeypots and does some basic data management (for instance, it merges and unifies the collected log files, see below).

[2] In the *eDonkey* network, peers are identified by a *clientID*, which is their IP address if they are directly reachable (high ID) or a 24 bits number otherwise (low ID).

[3] In the *eDonkey* network, files are identified by their *fileID* or *hash*. It is generated from the file's content, which ensures that two files with the same content are considered as identical, regardless of their names.

B. Honeypots.

After receiving an order from the manager to advertise a file F , the honeypot adds this file to its shared file list. The *eDonkey* server then adds the honeypot to its list of providers for file F. A honeypot may therefore afterward be contacted by other peers looking for F. Notice that, due to peer exchange [18] and other high-level *eDonkey* features, the honeypot may be contacted by peers which are not connected to the server.

Each honeypot constructs a log file of all the queries it receives. The log file can be written directly on a hard disk or sent via network to the monitor.

Before going any further, we specify that, for each query, the honeypot saves the information contained in the *eDonkey* protocol concerning the message type, as well as metadata such as the IP address, port, name, *userID*[4], version of client and ID status (high or low) of the peers sending the queries; moreover, it collects information concerning the server (name, IP, port), as well as data concerning the network environment (such as the timestamps marking the reception of the packets by the system).

Once a peer is connected to a honeypot, the list and description of all shared files in its cache content are retrieved. Note that this feature is not available on all peers, as it can be disabled by the user.

A honeypot must pass for a normal peer on the network. For this purpose, we have modified the open-source *eDonkey* client Amule [19] so that it meets our needs. We detail the main modifications below.

File display. In the normal course of events, if a client has files to offer (the client application considers that all files belonging to a given directory are files to be shared with other users), an OFFER-FILES message describing these files is sent immediately after the connection to the server is established, or whenever the client's shared file list changes. This message can also be used as a *keep-alive* message, sent periodically to the server.
As our honeypots do not have any file in their shared file folder, we have added a module to change the shared file list, so that the honeypot automatically sends the desired OFFER-FILES messages to the server.

Communication with other peers. When a honeypot is connected to a server and advertises some files, it may be contacted by other peers wanting to download one of these files. The normal message exchange process between a peer wanting to download a file and a peer having the file is presented in Figure 1. The downloading peer tries to establish a connection with a HELLO message, to which the provider peer answers with a HELLO-ANSWER message. The downloading peer sends a START-UPLOAD query, declaring which file it is interested in; the provider replies with

[4] The user ID (also called user hash) is unique and is used to identify a client across sessions,however,the client ID is valid only through a client's session with a specific server [6].

an ACCEPT-UPLOAD message. The downloading peer then starts to query file parts and the provider sends the queried parts.

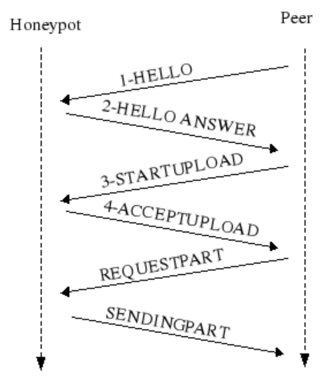

Fig. 1. Series of messages exchanged between honeypots and peers.

Our honeypots behave like provider peers up to the last part of the exchange: they do not send parts of the desired file to the downloading peer. In this aspect, honeypots do therefore not act as normal peers in the system, and run the risk of being noticed and then blacklisted. To avoid this, we have implemented two different strategies for answering file parts queries. The first one consists in sending nothing: the corresponding honeypots do not reply to file parts queries. The other strategy consists in sending random content when queried for file parts.

Log file construction. Our goal is to record users' activity concerning the files a honeypot advertises. We have therefore modified the application to record the following message types received from other peers: HELLO, START-UPLOAD, and REQUEST-PART.

C. Privacy concerns

For ethical and legal reasons, we cannot record any sensitive data concerning users' privacy. In our context, the main such data are IP addresses and filenames. We want however to be able to know if a same user requested several files or not.

In order to attain a high level of anonymisation, while keeping the anonymised data coherent, we follow a two-steps procedure. First, each honeypot encodes IP addresses in its log using a one way hash function, cryptographically sure. This anonymisation takes place before any data is written to disk or sent to the manager. This is however not sufficient to achieve a secure anonymisation of IP addresses: somebody could apply the hash function to all 2^{32} possible IP addresses, to construct a reverse dictionary of the anonymisation. After having collected the data from the honeypots, the manager therefore conducts a second anonymisation step: it replaces each hash value, in a coherent way between honeypots' logs, by an integer: the first hash is replaced by 0, the second one by 1, and so on. This ensures that the final anonymised data is secure, and that it is not possible to obtain the users' IP addresses from it.

In addition, we also anonymise file names which may contain personal information [20], [21]. We replace each word that appears less often than a given threshold by an integer.

IV. Experiments.

Many parameters may have a strong impact on the measurements conducted with our tool: which files the honeypot claims to have; number of honeypots in a distributed measurement; number of files advertised by the honeypot; duration of the measurement; etc.

In this section, we study several measurements which we have conducted to illustrate what is feasible with our approach and tool, and to investigate the impact of some key parameters. This relies on two different measurements:

- The *distributed* measurement used 24 honeypots ran by different PlanetLab [22] machines during one month (October 2008). Among the 24 honeypots, half did not answer at all to queries received from other peers; the others sent files with random content to the peers contacting them (see Section B below). All honeypots advertised the same four files (a movie, a song, a linux distribution and a text). They were all connected to the same large server and had a *HighID*.
- The *greedy* measurement used only one honeypot but aimed at advertising as many files as possible. To do so, the honeypot ran a two-steps procedure: during the first day of measurement, it asked their list of shared files to all peers contacting it, and added all these files to its own list of shared files; after the first day, it did not increase its list of shared files anymore, and just recorded the queries it received and the lists of files shared by peers contacting it. It did not send any content to other peers. We ran this measurement during the two first weeks of November 2008.

The key properties of the obtained data are summarized in Table I. These statistics already show that our measurement method succeeds in collecting large amounts of data, with hundreds of thousands distinct peers and files observed.

	distributed	**greedy**
Number of honeypots	24	1
Duration in days	32	15
Number of shared files	4	3,175
Number of distinct peers	110,049	871,445
Number of distinct files	28,007	267,047
Space used by distinct files	9 TB	90 TB

TABLE I
Basic statistics regarding the data collected with our measurements.

In the following, we investigate the impact of four key parameters of the measurement: its duration; the fact that honeypots send random content or no content at all; the number of honeypots involved; and the number of advertised files.

A. Impact of measurement duration.

Let us first observe how the number of distinct observed peers evolves as the duration of the measurement grows, displayed in Figures 2 and 3 for the distributed and greedy measurements respectively. It appears clearly that the number of observed peers grows rapidly, and linearily, during all the measurement. Even after 30 (resp. 15) days, the number of distinct peers observed by the distributed (resp. greedy) measurement still grows significantly: more than $2,500$ (resp. $50,000$) new peers per day.

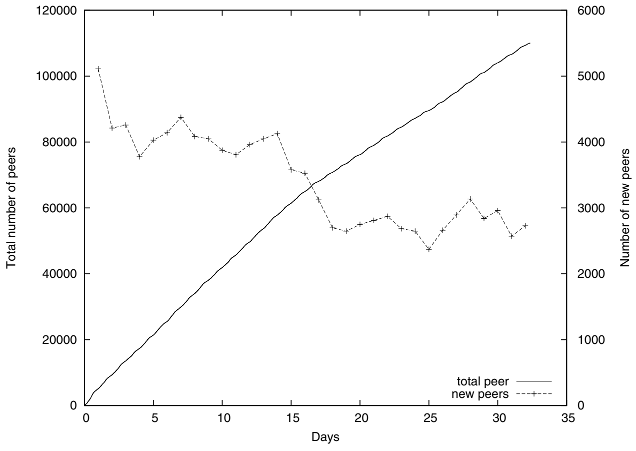

Fig. 2. Evolution of the number of distinct peers observed during our distributed measurement (left vertical axis) and number of new peers observed each day (right vertical axis) as a function of time elapsed since the beginning of this measurement, in days (horizontal axis).

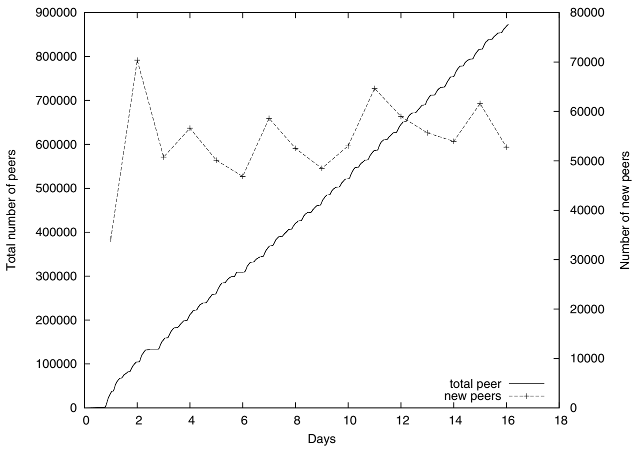

Fig. 3. Evolution of the number of distinct peers observed during our greedy measurement (left vertical axis) and number of new peers observed each day (right vertical axis) as a function of time elapsed since the beginning of this measurement, in days (horizontal axis).

These observations are of prime importance for conducting measurements: they show that conducting very long measurements is relevant, as one continuously discovers a significant amount of new peers. They also show that blacklisting, even if it is present, does not prevent us from observing many peers despite the fact that our honeypots never provide any useful content. We deepen this in the next section.

The number of new peers discovered each day is also displayed in Figures 2 and 3. It decreases during time in the distributed measurement, which is probably due to the fact that the popularity of shared files decreases. This may also be due to the fact that we reach a situation where most peers interested in the files proposed by the honeypot already have contacted it. In any case, the important point here is that this happens only after a very long measurement duration (one month), and that even then the number of new observed peers remains large (more than $2,500$ per day). This shows that continuing the measurement for long periods of time makes sense, even with 24 distributed honeypots advertising only 4 files. In such a scenario, one may have guessed that all the peers potentially interested in the proposed files, or most of them, would have been observed before 30 days. Of course, this depends on the popularity of the proposed files, but deepening this is out of the scope of this paper.

One may also notice that the number of peers observed during the first day of our greedy measurement (Figure 3) is very low; this is due to the initialisation phase of this measurement strategy, as described above: during the first day, the honeypot mainly constructs its large list of shared files, starting with only a few. The number of observed peers during this period of time is non-zero, but it is much smaller than after the initialisation phase and thus it is not visible on the plot. After this initial period, the honeypot observes an average of $54,000$ new peers each day, which is stable during the 15 days of measurement.

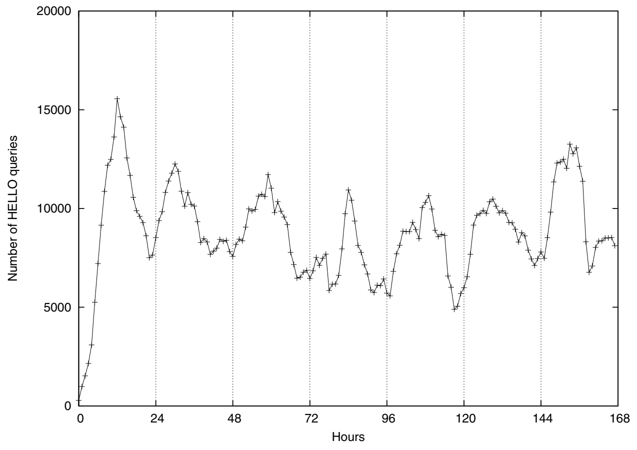

Fig. 4. Number of HELLO messages received each hour during the first week of our distributed measurements (vertical axis) as a function of time elapsed since the beginning of the measurement, in hours (horizontal axis).

Finally, we display in Figure 4 the number of queries we received during each hour of the first week of our distributed measurement (which is representative of other weeks and other measurements). After an initial phase in which our honeypots are not well known in the system (it lasts 10 minutes before we get our first query), a clear day-night effect appears. Such an effect reflects a regional nature of *eDonkey* activity: if observed users were scattered worldwide, such a day-night phenomenon would not appear. As these day-night variations follow the European and North African daily life variations (maximal values are reached during daytime in these regions and minimal ones during night), the plot may be seen as a confirmation of the well known fact that the *eDonkey* system is most popular in these regions [23], [24]. This may also be a consequence of a regional interest for the files proposed by the honeypot (for instance, music and movies are subject to such effects).

B. Random content vs no content.

When contacted by peers wanting an advertised file, honeypots may apply two different strategies: they may simply ignore these queries and not answer them; or they may send random content. These strategies may play an important role in avoiding blacklisting at server and/or peer levels: if the system detects that honeypots do not provide relevant content, then other peers may stop contacting them. We do not consider the strategy consisting in providing the true files, which would raise bandwidth and storage problems, as well as legal and ethical issues in many cases.

In order to investigate this, half the honeypots in our distributed measurement applied the first strategy, and half applied the second one. This leads to two groups of 12 honeypots, which we call *no-content* and *random-content*, respectively. Figures 5 and 6 show the number of distinct peers sending HELLO and START-UPLOAD messages (see Figure 1) observed by each group during our measurement. Similarily, Figure 7 displays the number of REQUEST-PART messages they received. Similar plots are obtained for each file advertised by our honeypots, independently (recall that the honeypots of this measurement setup advertised 4 different files).

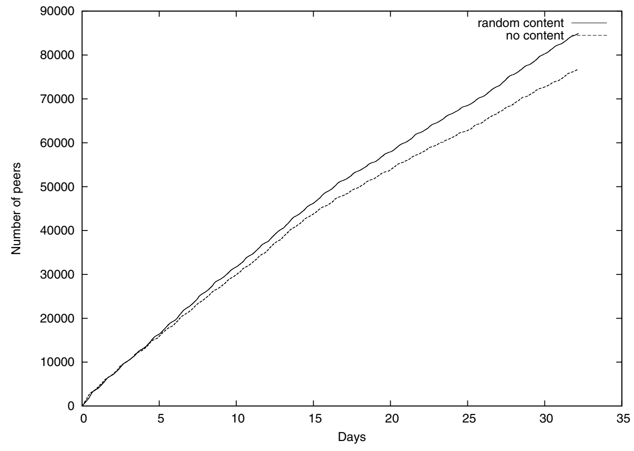

Fig. 5. Number of distinct peers sending HELLO messages to our two groups of honeypots in our distributed measurements (vertical axis) as a function of time elapsed since the beginning of measurement, in days (horizontal axis).

All these plots clearly show that, although the difference is not huge, the *random-content* strategy leads to better results than the *no-content* one. This is particularly striking

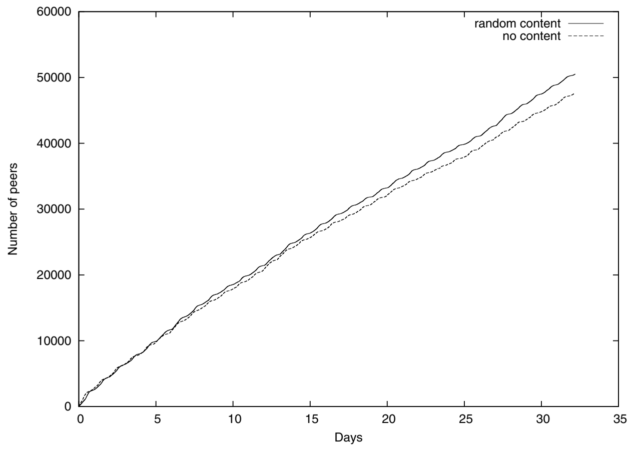

Fig. 6. Number of distinct peers sending START-UPLOAD messages to our two groups of honeypots in our distributed measurements (vertical axis) as a function of time elapsed since the beginning of measurement, in days (horizontal axis).

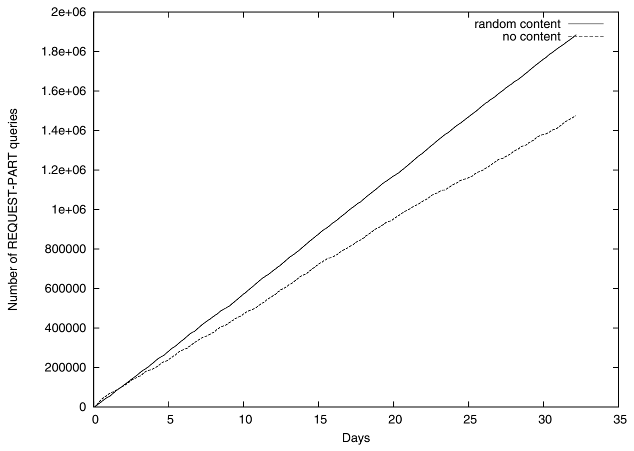

Fig. 7. Number of REQUEST-PART messages received by our two groups of honeypots in our distributed measurements (vertical axis) as a function of time elapsed since the beginning of measurement, in days (horizontal axis).

regarding the HELLO and START-UPLOAD messages, as the two strategies behave exactly in the the same up till this point. This demonstrates that, even though we still discover many new peers, there is some kind of blacklisting of honeypots, and that this blacklisting is more efficient when the honeypot sends no content. This is probably due to the fact that detecting honeypots which send invalid content takes more time than detecting honeypots which send nothing.

The larger difference between strategies regarding the number of REQUEST-PART messages, as observed in Figure 7 (we finally observe $1,9$ million queries with the random-content strategy, and only $1,5$ million with the no-content strategy), may be explained as follows. First, when a peer receives irrelevant data or no data from our honeypots, it may stop using it. Detecting the fact that the honeypot sends random content is also longer than detecting that it does not answer, and thus a peer may send queries for more file parts in this case before deciding not to consider the honeypot anymore. This may be seen as an implicit blacklisting at client level, which stops using a honeypot after it observed that it sends no useful content.

Finally, these plots show that the *random-content* strategy is significantly more efficient than the *no-content* one. Other blacklisting techniques may operate, but we cannot observe them with our data.

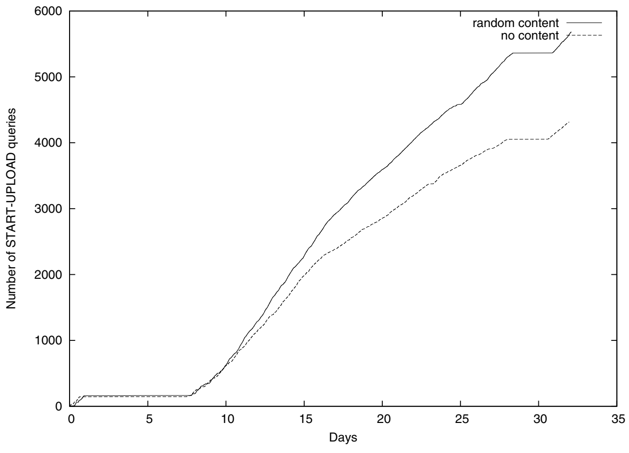

Fig. 8. Number of START-UPLOAD messages received *from a single peer* by our two groups of honeypots in our distributed measurements (vertical axis) as a function of time elapsed since the beginning of measurement, in days (horizontal axis).

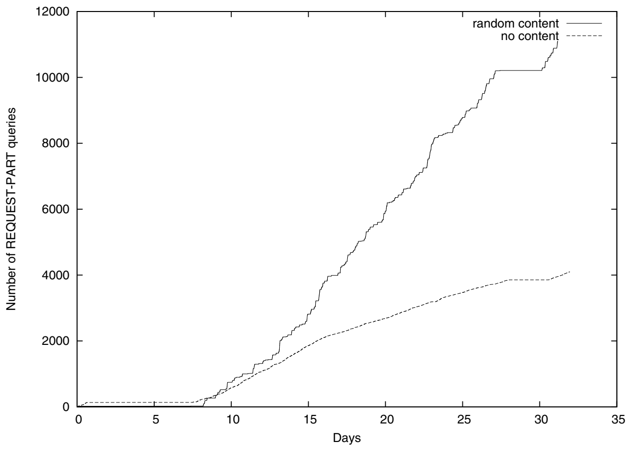

Fig. 9. Number of REQUEST-PART messages received *from a single peer* by our two groups of honeypots in our distributed measurements (vertical axis) as a function of time elapsed since the beginning of measurement, in days (horizontal axis).

In order to investigate further the difference between the *random-content* and *no-content* strategies, we plot in Figure 8 (resp. Figure 9) the number of START-UPLOAD (resp. REQUEST-PART) messages received by our two groups of honeypots from a *single* peer. The plot for HELLO messages is very similar to Figure 8 so we do not reproduce it here.

The peer we have chosen is the one which sent the largest number of queries to our honeypots. For some periods of time, it does not send any queries (which induces a plateau on the plots), but in general this peer sends queries as fast as it can, provided that the previous query was finished before it sends the next one.

The fact that it sends significantly less queries to our *no-content* honeypots than to our *random-content* ones confirms that the queries which receive no answer are sent at a lower rate than the ones which receive random content, as claimed above. This also explains the fact that the plot for *no-content* is smoother than the one for *random-content*, see Figure 9: the time between two queries to a *no-content* honeypot is constant (it is the timeout of the peer waiting for an answer), while for *random-content* this time may vary.

Notice however that this is not sufficient to explain the difference between the two strategies observed in Figures 5 and 6, as they display the number of distinct peers (not messages) observed. As explained above, this is certainly due to a combination of some kind of client-level blacklisting combined to the difference of speed at which a query is processed when sent to *no-content* and *random-content* honeypots.

C. Impact of the number of honeypots.

Our tools makes it possible to conduct honeypot measurements from a large number of machines at the same time, in a distributed manner. In this section, we explore the benefit of this feature: one may imagine that increasing the number of honeypots does not improve the measurement, at least as soon as a quite small number of honeypots are in place. The key question we want to address here therefore is: given a number n of honeypots, what is the benefit of adding one more honeypot to the infrastructure.

To investigate this, we use our distributed measurement involving 24 honeypots, and explore what we would have seen if only a part of them were used. To do so, we select n random honeypots among the 24 ones, for n between 0 and 24, and we compute the number of distinct peers observed by these n honeypots alone. As the choice of the n honeypots may have a significant influence on the result, we repeat this computation 100 times (we would ideally consider *all* 2^{24} possible subsets of our honeypots, but this is not feasible), and we plot the average, maximal, and minimal obtained values, see Figure 10. The average makes it possible to observe what one may expect. The maximal and minimal values give an indication of the dispersion of obtained results. In this case, they are quite close to the average, except at the very beginning of the plot (a honeypot leads to the observation of as many as $37,000$ peers, while another only sees $13,000$).

This plot clearly shows that there is an important benefit in using a few honeypots rather than only one. It also shows that, even when 24 honeypots are used, there is a significant benefit in adding more honeypots, thus calling for the use of large-scale distributed infrastructures. However, the benefit obtained by using more honeypots progressively decreases. This indicates that, even though many more honeypots may be used, at some reasonable point the benefit will become very low. Exploring this further requires distributed measurements from many more

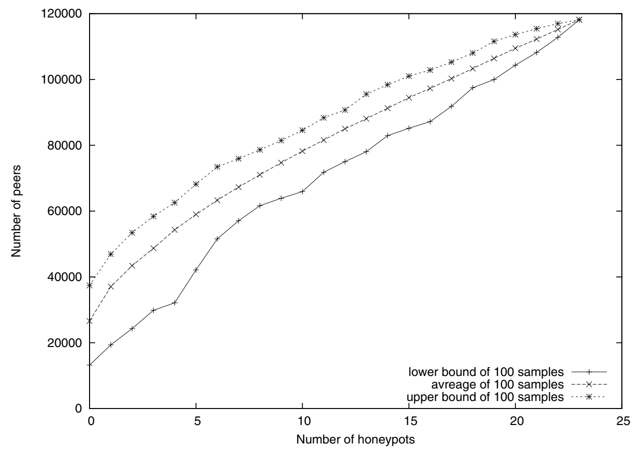

Fig. 10. Number of distinct peers observed at the end of the measurement (vertical axis) as a function of the number n of involved honeypots (horizontal axis). For each n, we sample 100 random sets of n honeypots and plot the average, minimal and maximal obtained values.

machines, which we leave for future work.

D. Impact of the number of files.

A natural way to increase the number of peers observed by our honeypot measurements is to increase the number of files we advertise, thus reducing the focus of the measurement. This is the aim of our *greedy* measurement, which starts by collecting for one day the list of files shared by the peers contacting a honeypot, and then adds all these files to the list of files shared by the honeypot, as described above.

Similarly to the previous section, we consider a measurement with a given number of files, and then, for any n lower than or equal to this number, we study the number of peers we would have observed if we used only n of these files in our measurement. Here, we could in principle consider the 3,175 files advertised in our measurement. However, this is not feasible in practice; we therefore consider two subsets of files: the *random-files* are a set of 100 randomly chosen files; and the *popular-files* are the 100 files for which we received queries from the largest number of peers.

For these two sets of files, we select n random ones among the 100 in the set, for n between 0 and 100, and we compute the number of distinct peers sending queries for at least one of these files. As the choice of the n files may have a significant influence on the result, we repeat this computation 100 times (we would ideally consider *all* 2^{100} possible subsets of files, or even the $2^{3\,175}$ ones, but this is not tractable), and we plot the average, maximal, and minimal obtained values, see Figures 11 and 12.

In both cases, the average, minimal and maximal values behave similarly, but the difference in the number of peers observed at the end of the measurement (100,000 for *random-files* and 270,000 for *popular-files*) clearly shows that the advertised files have a significant impact on the obtained data. This is confirmed for any number of files; in particular, the most popular file leads to the discovery of 13,373 peers, while some files lead to the observation of 2 peers only.

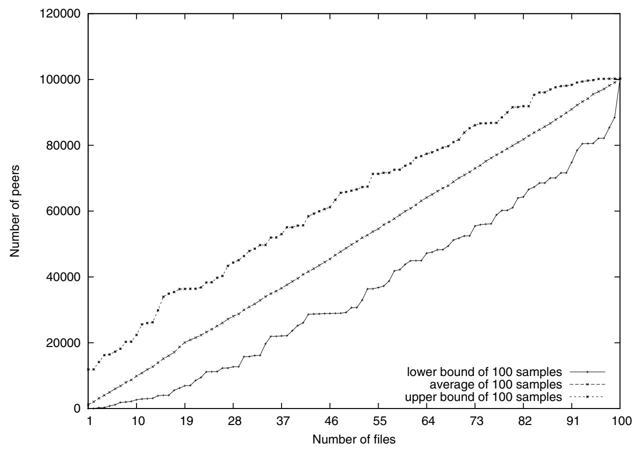

Fig. 11. Number of distinct peers observed at the end of the greedy measurement (vertical axis) as a function of the number n of advertised files in the *random-files* set (horizontal axis). For each n, we sample 100 random sets of n files and plot the average, minimal and maximal obtained values. In average, each new file leads to the discovery of approximately 1,000 new peers.

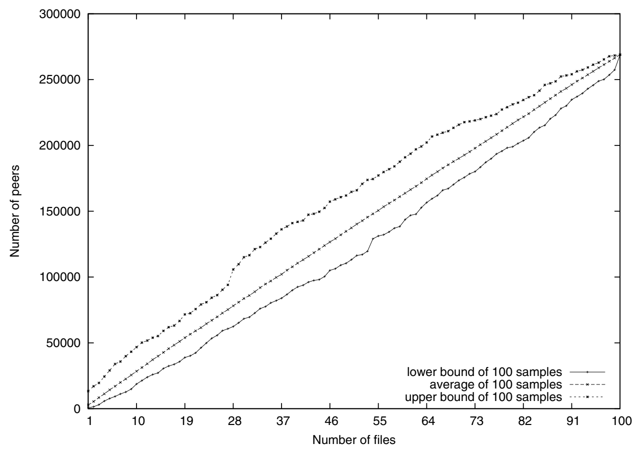

Fig. 12. Number of distinct peers observed at the end of the greedy measurement (vertical axis) as a function of the number n of advertised files in the *popular-files* set (horizontal axis). For each n, we sample 100 random sets of n files and plot the average, minimal and maximal obtained values. In average, each new file leads to the discovery of approximately 2,700 new peers.

Importantly, these plots also show that the number of peers increases significantly and linearily as we add more files to our shared files list. This shows that the greedy approach certainly is relevant if one aims at observing huge number of peers, and that advertising more than 100 files definitively is relevant.

V. Conclusion.

We have described a method for peer-to-peer measurement via honeypots and implemented it on the *eDonkey* network. This method makes it possible to collect large amounts of data on peers searching for files. It complements other approaches and has its own advantages and drawbacks: it makes it possible to focus on a topic of interest, or even a specific file; it does not require cooperation with a server administrator and/or an ISP; but it gives a very partial view; and it may interfere with the systems as it is active.

We have illustrated the use of our approach by conducting some measurements which show that it is relevant: it captures information on many peers and files (hundreds of thousands) during long periods of time (several weeks), and using a relatively large number of distributed honeypots (24).

Using these measurements, we have evaluated several strategies and parameters. First, we studied the impact of measurement duration, and showed that it is useful to make very long measurements, as we continuously observe new peers and files. Then we compared the results obtained by sending random content to other peers *vs* not answering to their queries; there is no huge difference between the two, but sending random content is more efficient. We then studied the impact of the number of honeypots involved in a measurement, and the impact of the number of files they advertise; we showed that increasing both makes measurements much more efficient in terms of the number of observed peers and files.

Finally, the global conclusion is that it is relevant to conduct large-scale measurements in terms of both duration, number of involved honeypots, and number of advertised files. Presented results may also be seen as giving hints in the setting up of measurements using our infrastructure, depending on one's priority and available resources.

The most immediate perspective for future work therefore is to conduct measurements at a significantly larger scale. Another key direction is the design of a measurements targeting specific properties, like for instance being able to capture *all* the activity regarding a particular file or a set of files, and/or a specific keyword. The results presented here show that this is a difficult task: measurements advertising only 4 files and ran on 24 honeypots during one month do capture much data, but they do not reach a steady state.

More generally, designing strategies aimed at capturing different kinds of information is an important perspective: which files should honeypots advertise? how should distributed honeypots be coordinated? how many of them are necessary? ...

Analysis of the obtained data should also be deepened. In particular, we plan to explore the relationships between peers inferred from the fact that they are interested in the same files, and conversely study relations between files from the fact that they are downloaded by the same peers.

Acknowledgements. This work is supported in part by the European MAPAP SIP-2006-PP-221003 and French ANR MAPE projects.

References

[1] "Managing peer-to-peer traffic with cisco service control technology," Cisco, Tech. Rep.

[2] L. Plissonneau, J.-L. Costeux, and P. Brown, "Analysis of peer-to-peer traffic on adsl," in *PAM*, 2005, pp. 69–82.

[3] N. Leibowitz, M. Ripeanu, and A. Wierzbicki, "Deconstructing the kazaa network," in *WIAPP '03: Proceedings of the The Third IEEE Workshop on Internet Applications*. Washington, DC, USA: IEEE Computer Society, 2003, p. 112.

[4] W. Acosta and S. Chandra, "Trace driven analysis of the long term evolution of gnutella peer-to-peer traffic," in *PAM*, 2007, pp. 42–51.

[5] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 219–232, 2004.

[6] Y. Kulbak and D. Bickson, "The emule protocol specification," School of Computer Science and Engineering The Hebrew University of Jerusalem, Tech. Rep., January 2005.

[7] J.-L. Guillaume, M. Latapy, and S. Le-Blond, "Statistical analysis of a p2p query graph based on degrees and their time-evolution," in *IWDC*, 2004, pp. 126–137.

[8] S. Le-Blond, J.-L. Guillaume, and M. Latapy, "Clustering in p2p exchanges and consequences on performances," in *IPTPS*, 2005, pp. 193–204.

[9] F. Aidouni, M. Latapy, and C. Magnien, "Ten weeks in the life of an edonkey server," in *Proceedings of HotP2P'09*, 2009.

[10] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, 2000.

[11] D. Hughes, J. Walkerdine, G. Coulson, and S. Gibson, "Peer-to-peer: Is deviant behavior the norm on p2p file-sharing networks?" *IEEE Distributed Systems Online*, vol. 7, no. 2, 2006.

[12] D. Zeinalipour-yazti and T. Folias, "Quantitative analysis of the gnutella network traffic," Department of Computer Science University of California, Tech. Rep., 2002.

[13] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," 2002.

[14] S. B. Handurukande, A.-M. Kermarrec, F. Le Fessant, L. Massoulié, and S. Patarin, "Peer sharing behaviour in the edonkey network, and implications for the design of server-less file sharing systems," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 4, pp. 359–371, 2006.

[15] F. Le Fessant, S. Handurukande, A. M. Kermarrec, and L. Massoulie, "Clustering in peer-to-peer file sharing workloads," in *3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, San Diego, CA, February 2004.

[16] W. Saddi and F. Guillemin, "Measurement based modeling of edonkey peer-to-peer file sharing system," *Managing Traffic Performance in Converged Networks*, pp. 974–985, 2007.

[17] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy, "Transport layer identification of p2p traffic," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2004, pp. 121–134.

[18] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht," *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–12, April 2006.

[19] aMule: `http://www.amule.org/`.

[20] M. Alllman and V. Paxson, "Issues and etiquette concerning use of shared measurement data," in *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007, pp. 135–140.

[21] E. Adar, "User 4xxxxx9: Anonymizing query logs," in *Workshop on Query Log Analysis at the 16th World Wide Web Conference*, 2007.

[22] planet-lab: `http://www.planet-lab.org/`.

[23] N. Ben-Azzouna, F. Clerot, C. Fricker, and F. Guillemin, "A flow-based approach to modeling adsl traffic on an ip backbone link," *Annals of Telecommunications*, vol. 59, pp. 1260–1299, 2004.

[24] G. Haßlinger, "Isp platforms under a heavy peer-to-peer workload," in *Peer-to-Peer Systems and Applications*, 2005, pp. 369–381.