Controlling P2P Applications via Address Harvesting: The Skype Story

Anat Bremler-Barr
Interdisciplinary Center Herzliya
bremler@idc.ac.il

Omer Dekel
Interdisciplinary Center, Herzliya, Israel
dekel.omer@idc.ac.il

Ran Goldschmidt
Interdisciplinary Center, Herzliya, Israel
goldschmidt.ran@idc.ac.il

Hanoch Levy
ETH Zurich, Switzerland
hanoch@tik.ee.ethz.ch *

ABSTRACT

P2P applications have become a dominant force in the Internet, both in economical value and in traffic volumes. A "battle of power" is ongoing between the application providers and the Internet Service Providers (ISPs) on who would control this traffic. This is motivated by both economical incentives and QoS objectives.

Little is known to the ISPs about the architecture of such applications or on the identity of their sessions; these are hidden by the application providers (assisted by their distributed control structure) making the ISPs' life harder.

We are interested in Skype, as a very popular representative of distributed control P2P applications. We explore the possibility of getting control/blocking Skype sessions by harvesting its Super Nodes (SNs), which form the heart of the system, and by blocking the network clients from connecting to them. Using experimental results and an analytic model we show that it is possible to collect a large enough number of SNs as to block, with probability higher than 95%, the service for an arbitrary connecting client.

Using 71 machines over the course of 4 days, we collected over 107,000 SNs. We demonstrate that this process converges and discovers the vast majority of the currently active SNs. Analyzing the discovered SNs we found that 46% of them belong to dynamic networks, and hence the harvesting process is bound to be an ongoing one, learning each time the current IP addresses of the SNs. Additional properties of Skype's SNs were observed.

The results derived and the vulnerability to SN harvesting, though discussed in the context of Skype, are general and may hold for other Super Node based P2P systems.

1. INTRODUCTION

P2P applications have become a dominant force in the Internet, both in economical value and in traffic volumes. A "battle of power" is ongoing between the application providers and the Internet Service Providers (ISPs) on who would control this traffic. This is motivated by both economical incentives and QoS objectives.

Little is known to the ISPs about the architecture of such applications or on the identity of their sessions; these are hidden by the application providers (assisted by their distributed control structure) to make the ISPs' life harder.

A very popular representative of distributed P2P applications is SKYPE, on which we focus this work. Skype has revolutionized the way we make VoIP, IM and video communications over the internet – reliably, simply and for free. Skype's popularity is growing with more than 579M downloads, 196M registered users and over 9M active users concurrently [1]. Skype's founders are those who founded Kaaza [2] that revolutionized file sharing, using a P2P network [3].

Despite its massive popularity, little is known about Skype's inner-workings. Skype is a closed-source application and consequently, Skype Ltd. does not disclose its protocols and architecture. Extensive studies [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] were conducted to disclose the Skype architecture, protocols and inner-workings by using reverse engineering and measurement techniques. Still, many of the details are not known. Due to the closed nature of the Skype network, the Skype protocol can be viewed by network administrators as a black-box. This poses new challenges for the administrators. Skype's traffic is encrypted, and thus prevents the enterprise from monitoring employee communications, an important means in the fight against intellectual property theft and industrial espionage. Moreover, some corporations may wish to filter Skype in order to receive revenues from using other VoIP solutions, not offered gratis. The network administrators find themselves powerless against the Skype application which can easily bypass the network's main filtering tool - the firewall. Skype port usage is random and has the ability to also use the standard http and https ports which are

*On leave of absence from Tel-Aviv University.

rarely blocked.

There are currently several commercial products which offer the ability to filter Skype communication and several research papers on the issue [4, 5, 6, 10]. All these methods are limited since they are dependant on a specific Skype version traffic signature. If Skype Ltd. wishes, it may easily alter these signatures to render these methods useless. Moreover, by nature, signature techniques suffer from high false-positive rate.

In this paper we propose and study a new approach that one could use for getting control of Skype traffic. The approach is not based on individual signatures, but rather on the general architecture of the system, and thus can be applied generically to other P2P applications of similar architecture.

The architectural property addressed by our approach is the reliance of the system on an array of *Super Nodes* (SN). These Super-Nodes are the heart of the Skype network, since they maintain an overlay network among themselves. Skype clients (Skype users) are allocated one (or a small number of) Super Nodes with which to associate, in order to receive the Skype service. Our methodology is based on harvesting the SN addresses and on blocking users from accessing those SNs.

Harvesting SN addresses can be done using an array of Skype clients (each of which can be downloaded from Skype easily), which we call *Harvesters*, and tracking the SN addresses they receive, or alternatively, the SN addresses they try to communicate to. The latter approach is *immune* to *communications encryption* by Skype and thus is very resilient. It is also generic and can apply to other Super Node based P2P systems.

We demonstrate that we are able to harvest, almost in real time, the majority of the currently active Skype Super Nodes. We further show, using experimental results (Section 4), that our methodology can block client connection with a 95% probability of success.

Our experimental study also reveals that a large fraction of the SNs (about 46%) used by Skype can be considered as *dynamic IP address nodes*, that is, their address changes (in order of days). As far as we know we are the first to point out the churn rate of Skype's SN addresses due to dynamic IP addresses. We study this property and the statistics of the life duration of an SN address (Section 5).

In addition to experimentation, we construct analytic models of the address harvesting process and of the blocking process. We start by forming a probabilistic model under the assumption that all SN addresses are *static* (Section 6). This study allows us to derive both exact and approximate results and to demonstrate that the approximate approach yields a very good approximation of the SN blocking probability.

We then (Section 7) turn to construct a *dynamic SN address* model, which is, naturally, more complicated. We therefore exploit the approximation technique, whose accuracy was demonstrated for the static model, and apply it to the dynamic model. The model is constructed for a *general distribution* of the "life" of an SN and thus is not limited to Skype. The model can be used to evaluate the blocking probability as a function of the system parameters, the number of harvesters and the address dynamics statistics.

We then (Section 8) use the analytic models for evaluating the harvesting process performance. We show that if the addresses are static then blocking rate of 100% can be achieved. If the addresses are not extremely dynamic then very high blocking rate (above 90%)[1] can be achieved with about 70 harvesters. Note that our experimentally measured churn (dynamics) rate of today's SN addresses falls into this category, implying that using harvesting could block Skype with high probability. We further use the model to explore possible strategies (or parameter settings) that can be adopted by Skype to maximize its resilience to blocking. The Skype system can adopt a strategy of releasing addresses to the clients in a "slow mode", which becomes very effective in preventing blocking when the addresses are dynamic (harvesters do not have enough time to harvest the address). Nonetheless, if in return one increases the number of harvesters, even in sub-linear rate (relatively to the Skype's slow down rate) then one can maintain very high blocking.

We conclude that address harvesting can be highly effective to drastically degrading the QoS offered by P2P applications. The results also shed some light about distributed control systems (like P2P systems): While it is commonly believed that such architectures are highly resilient, it turns out that they can become quite vulnerable when attacked by other distributed systems.

The structure of this paper is as follows: In Section 2 we provide an overview of Skype. In Section 3 we describe the address harvesting technique. In Section 4 we describe the results of the experiments conducted on the Skype system and the properties observed from these experiments. In Section 6 we construct a model for address harvesting and analyze the effectiveness (blocking probability) achieved by the harvesters. In Section 7 we conduct a similar analysis for systems with dynamic address SNs. In Section 8 we use the results derived in Section 7 to provide performance analysis of the harvesting system and to derive the effectiveness of it blocking Skype. Lastly, concluding remarks are given in Section 9.

2. SKYPE OVERVIEW

Originally, Skype was a free Internet Telephony ser-

[1]One should keep in mind that even lower blocking rates, e.g. 50%, would imply very severe QoS degradation for Skype

vice. Presently, it offers many other services free of charge including IM, offline messaging, VoIP, a Video chat service and file transfer; As time passes more and more services are being developed. Skype services are expanded (commercial services with cost) to interact with current telecommunications networks such as PSTN [16] and SMS [17].

Skype is a P2P overlay-network based upon a partially centralized architecture[15]. All participating nodes in the Skype network are divided into two categories: Super-Nodes (SNs) and ordinary Skype Clients (SCs). A user who installed the Skype application and connects to the Skype network becomes a node and is referred to as an SC. If the SC surpasses some set of publicly unknown thresholds (bandwidth, CPU usage , Available RAM ...), the SC will take upon itself more tasks and effectively act as a Skype server which is referred to as a Super-Node. When an SC becomes an SN the SC user is not notified of its status and there is no option (such as pressing a simple button) for disabling it. The SC issues queries through the SN. The SC connects to one SN (or a small number of SNs) with which it communicates.

The Super-Nodes maintain an overlay network among themselves. The SNs communicate among themselves in a way that every SN in the network has full knowledge or access to all available users and resources. All SCs must be authenticated against one of the Login Servers (LS) in order to be able to gain access to the Skype Network and this is done by the SN. The SN relays the login request, thus allowing the SC to connect to the network[2]. The set of LSs is one of the few (if not the only) components in the network which are central. When a client wishes to communicate (IM, Voice, file-transfer or Video) with another client, it queries its associated SN regarding the availability of the destination client, and its current IP. After retrieving the IP address of the SC destination, the client usually transfers the data directly to the destination client. However, if the SC is unable to communicate directly with another SC, then the SNs can also relay all communication, thus effectively bypassing firewalls and proxy-servers.

When an SC attempts to connect to the Skype network for the first time, it attempts to connect to one of the hard-coded (bootstrap) SNs. After the initial connection, an updated list of SNs is saved locally on the SC. This list contains up to 200 SN IP's and their access ports. From this point onwards, if the SC wishes to connect, it will attempt to connect to its default SN. If unsuccessful, it will fall back to other SNs in its list. If the SC cannot connect to any of the SNs in the list or to the bootstrap SN, it will not be able to connect to the Skype network.

The list of SN is updated continuously, we suspect this is done since skype handles the fact that some of the SNs become inactive with time (we analyze the reasons for this in Section 5). Hence skype updates the list from time to time and replaces the inactive SNs with active SNs. Using the fact that the SN list is not encrypted in Skype versions 2-2.5 we observe that on average of 6% of the list is changed per hour. However, if we remove the SN list, skype downloads a new list, where 75% of the SNs in the new list are new. In this paper we use this characteristic of the Skype login procedure to block the Skype application.

3. THE SN ADDRESSES HARVESTING TECHNIQUES

As explained in the previous section, if one can compile a list of SNs to which the SC can connect and block access to them - then one can block the SC from connecting to the Skype network altogether.

In order to compile a list of SNs we developed two methods for harvesting SNs from the Skype network: 1. Extracting the SN list 2. Monitoring Skype Connections. The first technique works only if the list is not encrypted, while the second technique works even if the list is encrypted.

We should note that for each SN we collected, both the IP and the port and used the pair as the SN identifier. This would serve to eliminate false-negatives of our blocking technique, namely prevent the accidental blocking of another service which uses the same IP address but uses a different port. From this point on, when we refer to the harvested SNs addresses, we refer to the pair (IP, port). Thus, an IP that has two different ports for Skype application, will be counted twice.

3.1 Extracting the SN list

In this technique the harvesting of the SNs is based on the fact that in Skype versions 2-2.5 each SC holds a list of up to 200 SNs and their connection port in a specific XML file ($\%appdata\%\backslash skype\backslash shared.xml$).

We harvested the Skype network for SNs by repeatedly doing the following on an SC: 1. Extracting the SN address and ports from the XML file; 2. Flushing most of the SN addresses from the list - leaving only specific SNs 3. Restarting the Skype client and waiting until the list is refreshed with 200 SN addresses. Each such iteration, takes approximately 2-2.5 minutes. While the technique is similar to the work of Guha et. al. the aim is different. Guha et. al. focuses on collecting the subset of SNs over long periods of time, and hence used only two computers to collect the SNs over a period of four months (therefore, their compiled list cannot be used

[2]Baset et al[4] present that the SC can attempt to directly authenticate itself against the LS, however, we did not observe this. This bears little relevance because even if the SC can authenticate directly against the LS, we can simply add the LS addresses to blocked list

to block skype). We focused on harvesting the entire set of active SNs, and hence we used clusters of dozens of computers all harvesting the SNs concurrently. Our aim was to receive an accurate snapshot of the SN Set.

In order to discover the bootstrap SNs of Skype, which do not necessary appear at the XML file, we observed the preliminary connection attempts of a newly installed SC. Using netstat and a sniffer application we discovered the addresses of the bootstrap SNs. Most of them had the same specific port. We added these SNs to the list of harvested SN.

3.2 Monitoring Skype Connections

An alternative method for SN IP harvesting is observing outbound connections (with the aid of the Netstat command or other network monitoring tools). With the command "netstat -b -n -o" the client's computer lists the active connections (IP,port) with an indication of the application that was responsible for opening the connection (in our case indicating that this is a skype connection). In order to force Skype Client to connect to another SN we block the SN with a local or external firewall. The skype client then switches to another SN, assuming this SN is not currently available. Note, that due to the nature of P2P, skype is designed with the assumption that some of the SNs would not be available, since they left the network - and hence has built-in fast mechanism to switch from one SN to another. After collecting around 200 SNs, we erase the SN list file - and thus ensure that the client receives a new pool of SNs. This technique works in all versions of skype, including the newest version, where the SN list is encrypted. [3]

4. EXPERIMENTAL RESULTS

In this section we show the feasibility of harvesting sufficient active SNs, in order to block skype with high probability. We focus on the first technique, extracting the SN list. We also did preliminary testing on the second technique, monitoring skype connections, and found that it is feasible to collect 30 new SNs per minute.

We implemented, the first technique, extracting the SN list, by using a cluster that consisted of 77 harvesters, where a *harvester* denotes a computer that collects the SN addresses. We note that one can also run different harvesters on the same machine, since skype supports running multiple clients on multiple users on the same machine [18] and by this reduced the cost of harvesting. [4]

To achieve some geographical diversity 73 of the harvesters were split across two separate sites (located in Israel), and 4 were placed in Zurich. The experiment was conducted over a period of 80 hours (approximately 2700 iterations).

Using our experiments the harvesters collected over 40 Million SN addresses (IP, port) pairs, and discovered over 107,000 unique SNs. There are only 106,300 unique SN IP addresses in the list. It is apparent that the difference is negligible, and almost all the SNs only use one port. Over the duration of the experiment (80 hours) 2700 iterations of harvesting were conducted. Figure 1 (a) shows the cumulative number of SNs discovered as a function of the number of harvesters used (one can control this number by disregarding the data collected by some of the harvesters). It is clear that the majority of the SNs, 100,000, were discovered by the first 30 harvesters. The last 41 harvesters discovered a total of roughly 7,000 SNs. This means that each of these 41 harvesters discovered 175 new SNs in its 2700 iterations. This means, that approximately each one of these harvesters discovered new SNs only in 6.6% of the iterations. This is a good indication for the convergence of our process and that it discovered most of the SNs.

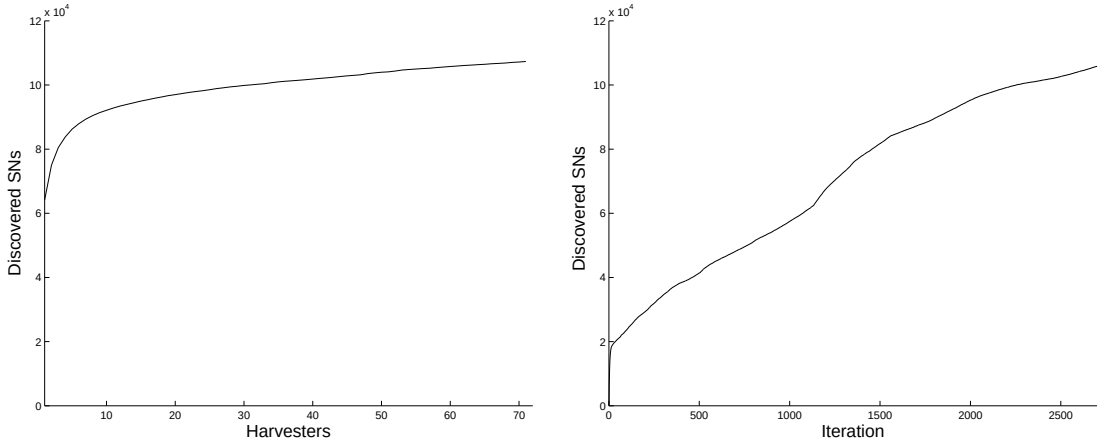
Figure 1 (b) shows the cumulative number of SNs discovered by the collection of the 71 harvesters as a function of the number of iterations. We can see that the harvesters consistently found new SNs at each iteration (though at a decreasing rate). We note that, as we explain in subsection 4.2, the majority of the SNs belong to dynamic IP networks. Hence, we conjecture that many of the new SNs represent a previously found SN whose IP address changed. We note that the big increase in the slope at 1200-1500 iterations, is at the normal office hours in the United States, which, as we show in the subsection 4.2, is the origin of 20% of the SNs.

4.1 Blocking Capabilities

In order to measure the probability of blocking an SC from connecting to Skype from within the Enterprise network, we ran 12 Tester Clients (TC) that simulate the attempts of a regular user to connect to Skype. Connection attempts from within the enterprise can be classified into 3 types. The first consists of a freshly installed SC. This case is easy to block, since it will attempt to contact only SNs from a hard-coded list, which, as explained, is fixed and thus easy to block. The second consists of an SC which was installed outside of the enterprise (an enterprise user who installed Skype at home on his portable computer) and returns to the enterprise network after a lengthy period (days and even hours). This is also an easy to block option since our experiments found that the harvesters discovered all "old" SN addresses after a short period of time. The third case is the most challenging one to block,

[3]We note that even without erasing the SN list one can receive a new pool of SNs but in a lower rate.

[4]We experimented with this technique and were able to run 15 independent clients on the same machine, however, we did not use this feature in our experiments.



(a) (b)

Figure 1: (a) The cumulative number of SNs discovered as a function of the number of harvesters (b) The cumulative number of SNs discovered as a function of the iteration number

and we focus on testing it: It consists of a user who retrieves a very 'fresh' list of SNs. This is a case of a user who installed the SC, connected and used it in some environment that allowed it to connect (home or an internet coffee shop). After connecting and refreshing the SN list, the user returned, after a very short time, to the enterprise network, which now aims at blocking Skype.

In order to simulate this case, our Tester Client performed rounds of 20 minutes, each consisting of the following steps by a TC: Start Skype, log in, stay idle for 10 minutes and shut down for 10 minutes. After each round we checked whether the client can connect to Skype, namely, whether the client's SN list contains an SN which was not yet discovered by the harvesters.

For each of the 12 TC's we performed 240 such rounds during 4 days. Figure 2 (a) depicts the fraction of the attempts (denoted as blocking probability) that could not successfully connect to the Skype network. This is calculated as the fraction of tests in which all the TC addresses were discovered earlier by the harvesters. We measured this probability of blocking immediately after the completion of the TC round, and 10 and 30 minutes after each round. These different scenarios aim at simulating realistic cases, representing the times it takes the user to return to the enterprise network and attempt connecting to Skype. Our results show that the average probability of blocking is very high, ranging from 94.5% to 95.5%. Figure 2 (b) depicts the probability of a TC to successfully connect as a function of the round number. Not surprisingly, the figure demonstrates an increase of the blocking probability in the round index. This is due to the fact the cumulative number of harvested addresses increases with the round number.

4.2 The Characteristics of the SuperNodes

Below we summarize some of the characteristics of the discovered SNs, emphasizing those characteristics which can be useful in designing the blocking mechanism:

SN Distribution: In the experiments we collected $2700 * 200 * 77 = 40M$ records. We counted the frequency of each SN within these records. In Figure 3 (a) we show a CDF of the frequency distribution where we count the relative frequency percentage of records in which the first most frequent SNs appear in. It appears that the 10% of the most frequent SNs cover roughly 80% of the total collected records.

Port Distribution: When Skype is installed, a port is chosen at random as the default port to be used. We checked the usage frequency of different ports. We analyzed all port usages and found that the most frequent port which appeared in 0.86% of the collected records, belonged to bootstrap SN. The most frequently used port which was *not used* by any of the bootstrap SNs was in 0.34% of the total discovered records.

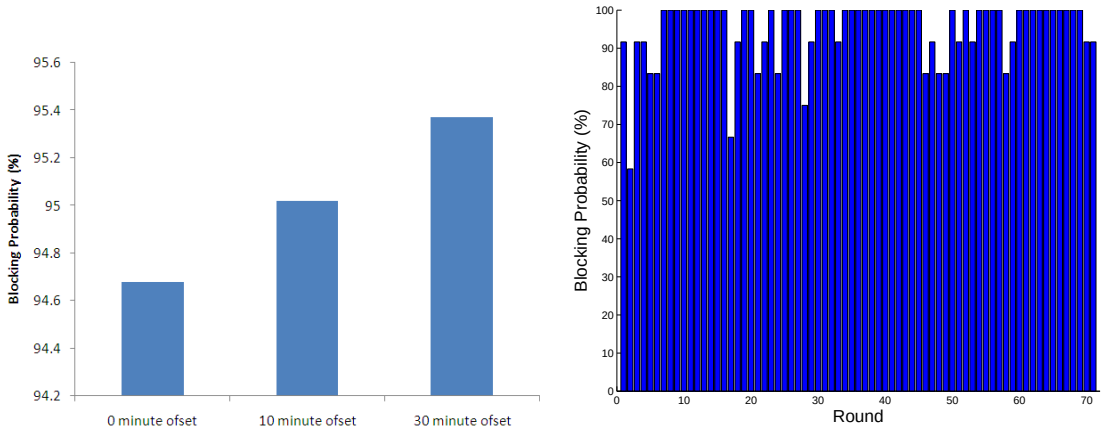
Geographic Distribution: We found no dominant AS. However, we observed that several countries are more dominant than others. Figure 3 (b) depicts distribution of SN by country. We note that the country distribution observed by the Israeli-located harvesters is similar to that observed by the Swiss-located harvesters.

Dynamic vs Static IP Distribution: We used the SORBS project Policy Block List[19] to query whether or not the IP addresses are dynamic.

We found that 46% of the IPs are dynamic. This means that in order to have current SN lists, one needs to repeatedly discover the SNs, since large portion of them change their IP over time.

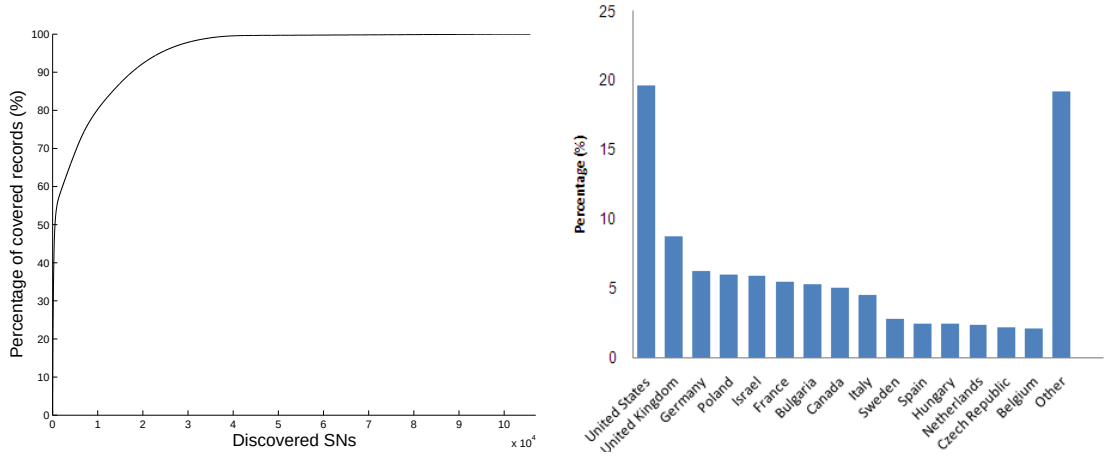
5. THE DYNAMIC NATURE OF SNS

In the next section, we model the Skype network, in order to model the effectiveness of the harvesting



(a) (b)

Figure 2: (a) The blocking probability 0, 10 and 30 minutes after each round (b) The Blocking probability as a function of the round



(a) (b)

Figure 3: (a) Appearance Distribution of Discovered SNs (b) Country Distribution of Discovered SNs

	From All SNs' IPs	Dead SNs' IPs
Total	100%	58.27%
Static	53.97%	50.6%
Dynamic	46.02%	66.9%

Table 1: The percentage of dead SNs' IPs after 21 days, classifying the IPs according to their address type

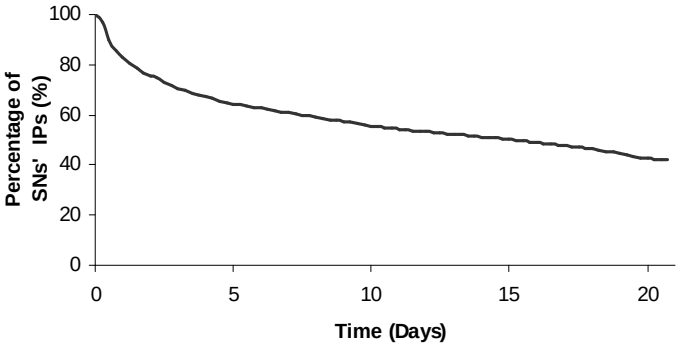


Figure 4: The percentage of SN IPs with residual life length $> x$

mechanism in blocking Skype. In order to do that we need to understand one more key issue about the SNs of Skype - its churn. The churn in P2P networks is very important and has been extensively studied [20, 21, 22, 23, 24, 9]. It measures the continuous change of nodes joining and leaving the network.

The churn of the SNs plays an important role in understanding the blocking probability of our harvesting technique. On the one hand, some of the SNs in the client list may be unavailable at the time the client tries to connect to skype (and this increases the blocking probability). On the other hand the harvester needs to repeatedly learn the current active SNs - since new SNs may join the network, which may make it difficult for the harvester to learn all the current SNs, and hence reduce the blocking probability.

The churn of skype results from numerous reasons:

1. Due to dynamic IP addresses used in various parts of the internet, an SN might change its address and thus it may appear as a new entity in the SN list. As we have shown before 46% of the IPs have a dynamic address, and hence after few days they change their IP. We argue below, that the dynamic nature of dynamic IPs is the most important reason for the churn of SNs . As far as we know we are the first to point out the important role of dynamic IP to the churn of skype. A similar phenomenon may occur for a different reason; if the computer changes its location (in the case of a laptop).

2. New skype clients, that become SNs and the opposite, the case of dropout clients that were SNs and left the network permanently.

3. The SNs that temporarily left and then rejoined the network , i.e. closing/opening the Skype client or, more likely, shutting down/ opening the computer. Note that in this case the harvester learned those IP addresses in the past.

4. The system may choose to abandon old SN's and nominate new ones.

Surprisingly, we did not find any research that can give us an exact model of the churn of skype. Guha et al[9], did extensive research on the churn of the Skype Super-Nodes, but it focuses only on the availability of Super-Nodes that had a long life span (over 3 months). The paper collected the Super-Nodes over a long period of 3 months, and then took a sample of the Super-Nodes and checked their availability. Hence the Super-Nodes they checked were those that were alive after the period of 3 months. Moreover they did not research the amount of new nodes that joined skype over time.

We give here an estimate of the number of new SNs' IPs and the dead SNs' IPs over time. An estimate of the number of new SNs' IPs that joined Skype per time unit can be learned from Figure 1 (b). At each iteration (2000-25000) on average 9 new SNs' IPs were discovered, where iteration as recalled takes approximately 2 minutes. Note, that since we saw the testers achieved high our blocking probability of around 95%, it is clear that most of them are new ones and not SNs that the harvester technique did not succeed in discovering. [5]

In order to understand the number of SNs's IP that died per time unit, we take a sample size 5026 from the discovered SNs' IPs at some specific time unit and start to ping them by mimicking the login connection of skype to an SN for a period of 21 days . The ping is an encrypted UDP packet, that we captured from a regular connection attempt of a skype client to an SN (similar to the technique of [9]) [6].

[5]Take for example a test that was done in iteration 1000. If we compare it to iteration 2000, we can see the harvester had learned in this period more than 40% of new SNs' IPs. However at the point of iteration 1000, the Tester had a blocking probability of 95% - hence most of the discovered SNs are new SNs' IPs that were not alive at the time of iteration 1000.

[6]In order to validate the technique, we compare the result of the UDP check to the the most accurate possible availability-check - using the client as a black box - and letting it check the availability by performing the full login process. Again we use the fact that in the old versions of Skype (2-2.5) the list of SNs is not encrypted. Hence we clear the list of SNs and replace it with the SN that we want to check its availability. We restart the client, and the client makes an attempt to login to the SN that we inserted for it. We then capture all the communication between the Skype Client and see if the skype received a response from the SN (ignoring RST responses). We found out that checking with the captured UDP packet (as compare to the black-box checking) is accurate in the vast majority of cases. Note that in the full login process of skype, skype checks also try to connect to the SN through a TCP connection with various

An SN's IP dies if the checked SN's IP at some time stops being available, and it remains unavailable until the end of the experiment. In checking if an SN's IP died we have difficulty in categorizing an SN's IP that starts suffering from unavailability only towards the end of the experiment. To overcome this we decided arbitrarily that a node is dead if it did not, at least, respond in the last three tests of the experiment.

Figure 4 shows the residual life of SNs' IPs. Approximately 18% of the SNs died in the first day, 8% in the second day. 5% in the third day and only 3% in the fourth day (total 34% of the SNs died in the first four days. After the first 4 days the number of SNs' IPs that died is more moderate, and is around 1.25% per day.

We speculate that the residual life of the SNs' IPs is less than 4 days for 34% of the SNs is related to the fact that some of the SNs belong to dynamic IP networks (usually home users connected by cable, xDSL and so on..). In this case the SN's IP died since the IP address of the SN has been replaced. However, there is a good chance that the SN is alive but with a different IP address. ISPs assign to the majority of the home users a dynamic IP address meaning that the each time the user connects he will not necessarily be assigned the same IP address as before. We note that dynamic IPs may occur also due to NAT or a cluster of proxies, however this is not relevant in the case of a Skype SN, since an SN is not behind a firewall or NAT [2, 9]. By querying the SORBS project[19] we characterized the address space that our SNs belong to. Table 1 summarizes the results. We discovered that 46.02% of the total checked SNs' IPs belong to dynamic IP networks. 66.9% of the dynamic IP SNs had died after 21 days of the experiment, while only 50.6% of the static IP SNs had died. However, this number is somewhat inconclusive. Since 33.1% of these dynamic IP SNs did not die.

We note that according to Xie et al[25] that carried out a massive check on dynamic IPs, the SORBS type of project is only 70% accurate, and hence the results may be inconclusive due to mistakes in classification. Nonetheless, this is the most accurate publicly available database - and hence we do not have an alternative and we rely on it to receive some understanding of the characteristics of the IPs.

The massive number of dynamic SNs' IPs that died during the first number of days, correlates with the median inter-user duration of a dynamic IP, presented by Xie et al. In the first 24 hours we lost 24% of the SNs with dynamic IPs (Xie et al estimated $\sim 30\%$) and in the following three days (day 2-4) we lost 18% of the SNs with dynamic IPs (Xie et al estimated 30%). Note, that also according to Xie et al part of the dynamic IPs have a long life span, and 8.5% of the dynamic IPs died between the third and the seventh day and 17% of the dynamic IPs died after the seventh day. This is due to the fact, that in some cases of dynamic IPs, the IP is replaced not according to some lease time limit (which is usually 3 days for dynamic IPs) but as the outcome of a disconnection of a computer from the network. Today many of the computers are always on-line and the change of IP in these cases rarely occur. We speculate that Skype, while choosing to transfer clients to SNs, prefers to choose from the groups of dynamic IPs with a long life span.

There are several implications due to the dynamic nature of Skype SNs on our harvesting technique. First, it implies that the process of harvesting is continuous - since the population of SNs' IPs changes with time. Second, we should be careful and note that the population size gathered in 4 days, 110K in our case, is dramatically different from the active population SNs in each time unit. Due to the dynamic nature of skype - an SN that change its IP in those 4 days will appear twice in our count. In order to estimate the active population in each time unit, we assume for simplicity that this size is fixed. We suspect that this is not accurate, but only gives a good enough approximation. We suspect that the number of SNs may change proportional to the number of active clients. Hence during regular working hours, the number of active SNs is higher.

We estimate the number of active SNs by assuming that the number of SNs that died in each iteration is equal to the number of new SNs discovered at each iteration. From figure 1(b) the number of new nodes that were discovered per iteration is around 9 (we take the number based on the last 2000-2500 iterations and the fact that the we reach a blocking probability of 95%), and the percentage of SN that died in each iteration is 0.000208 (according to the number of dead SNs in the first iteration of our test, where we test a representative sample of skype SNs close to the time the set was collected). Hence $M \cdot 0.000208 = 9$ where M is the active SNs per time unit. We can estimated M to be approximately 45K.

The third implication of the dynamic nature of the list, is the requirement for a garbage collector, that eliminates from the list IP addresses that are no longer active. This, in order to make sure the list of SNs' IPs will not blow up. We plan to investigate the design of such a garbage collector in future work.

REMARK 5.1. *One important indirect impact of our research in the general P2P area is a simple guideline of how to chose Super-Nodes that minimize the churn. The previous technique, presented by Guha et al[24], focuses on algorithms that take as input only the history of the Super-Nodes. We note, that a key impact of the churn of the a super-Nodes is its IP address type. Supernodes from static IP addresses have more chances of remaining alive for a longer time in the network, and*

ports [9].

hence should be preferred.

6. BLOCKING SKYPE: MODELING AND ANALYSIS OF A STATIC ADDRESS MODEL

Below we provide a simple model aiming at evaluating the effectiveness of the harvesting mechanism in blocking Skype. Our objective is to evaluate the probability of successfully blocking Skype as a function of the number of iterations preformed by the harvesters. This will yield an indication to the effectiveness and practicality of the methodology. Specifically, if too many iterations are required, it may lead to the infeasibility of the technique, mainly due to the fact that the SN population is dynamic and needs to be learned continuously.

Formally we define M to be the number of SNs (which are distinct objects) in the system. We estimate that M is in the order of $100,000$. Let N be the number of SN addresses in the list of a Skype client. In our case $N = 200$. We consider the following SN selection process: We randomly select (with uniform distribution) N distinct objects (from the set of M objects). We now return the N objects and again randomly select N distinct objects. We repeat this K times. We then conduct this at the $K+1$st time and ask what is the probability that each of the objects selected at the $K+1$st time was selected earlier. This will represent the probability that a real Skype user ($K+1$st) will be blocked as a result of the harvesting conducted by the K selections.

Note, that this is a somewhat oversimplified model of Skype for the following reasons: First, we assume that the SN population of Skype is fixed over time. That is, over time, no SN's are removed or added to the SN set. This is not true as we shows in Section 5; this assumption will be relaxed in the dynamic model in Section 7. Second, we assume that the selection is done using a uniform distribution. However, from Figure 3 (a) it is clear that this is not precise. In future work we plan to relax these assumptions and adapt the model.

6.1 Exact Analysis

Let $X(l,k)$, $k \geq 1$, $1 \leq l \leq M$, denote the probability that the number of distinct elements selected by the first k experiments equals l. Let $X(l,k|l')$ denote the probability of that same event, given that the number of distinct elements selected by the first $k-1$ experiments equals l'. Then we have: $X(N,1) = 1, X(l,1) = 0$ for $l' \leq l \leq M$ and for $0 \leq l - l' \leq N$:

$$X(l,k|l') = \begin{pmatrix} l' \\ N-(l-l') \end{pmatrix} \cdot \begin{pmatrix} M-l' \\ l-l' \end{pmatrix} / \begin{pmatrix} M \\ N \end{pmatrix} \quad (1)$$

Thus, $X(l,k) = \sum_{l'=max(0,l-N)}^{l} X(l,k|l')X(l',k-1)$ for $k = 2\ldots$

The equations above allow one to recursively compute $X(l,k)$. Since each element $X(i,j)$ can be computed in $O(N)$ steps, the over all computation required is $O(NKM)$ calculations.

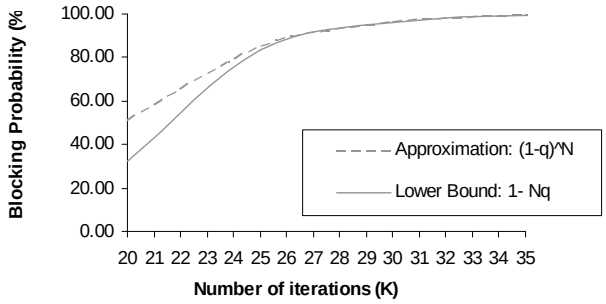


Figure 5: Comparison of lower bound and approximation equation or the static model with parameters N=50K, M=200, H=71

An alternative approach for computing $X(l,k)$ is to use an inclusive-exclusive approach as used in the baseball collector problem. The basic baseball card collector problem is: Given a set of M cards and a collector who receives k packs each consisting of N cards. What is the probability that the collector collects all M cards. This analysis is given in the appendix.

Now, we are interested in the probability that no new elements were selected at experiment $K+1$. To this end consider the collection of events where the number of distinct elements at experiment $k+1$ equals that of experiment k. The probability of this event, denoted by $B(k)$ is given by:

$$B(k) = \sum_{l=N}^{M} X(l,k|l)X(l,k-1) \quad (2)$$

and we are interested in $B(K+1)$.

6.2 Approximate Analysis

Since the exact analysis is provided by a recursion, we provide here an approximate closed form analysis that can provide an insight into the results and how they depend on the system parameters.

Consider address (object) x (1 of the 50,000). Let p be the probability that x is selected in iteration i. Since the experiments are independent of each other and of the history, then p is not a function of i. Then we have: $p = Pr[\ \ x \textit{ is selected on } i] = N/M = 200/50,000 = 0.004$ and $Pr[\ \ x \textit{ is not selected on } i] = 1-p$.

Now let $q = Pr[x \textit{ is not selected in experiments 1,...,K}]$. Since all experiments are independent of each other we have: $q = (1-p)^K = (1-N/M)^K$. Thus q is the probability that after K experiments x is still an unselected address. Now, we are interested in the probability that

a Skype client is blocked after conducting the Kth iteration. Thus, we are interested in conducting the $K+1$st iteration and in asking whether all N addresses of this iteration are already discovered addresses, i.e. blocked address.

Recall that for each of the N addresses received at the $K+1$st reading, the probability that this is a blocked address is given by $1-q$. To conduct the approximate analysis we will assume that for any two addresses $1 \leq i, j \leq N$ the events that these are already discovered addresses are independent of each other. This is not exact since these are dependent events.

Now consider a user who has $N(=200)$ addresses and ask what is the probability that all of them are blocked addresses and hence the user is blocked. Under the assumption of independence we get:

$$\begin{aligned} B &\approx Pr[\textit{all } N \textit{ addresses are blocked}] \\ &= (1-q)^N = (1-(1-N/M)^K)^N \end{aligned} \tag{3}$$

Since the above analysis is not exact - we provide a lower bound on B. To this end note that q is the probability that a single address is free (non blocked).

Let A_i be the event that address i $(i = 1, .., 200)$ is free. Let A be the event that either of the addresses is free. Then we have $Pr[A] = Pr[A_1 \cup A_2 \cup A_3 \cup \ldots A_n]$. Using Boole's inequality we get $Pr[A] = Pr[A_1 \cup A_2 \cup A_3 \cup \ldots A_n] \leq Pr[A_1] + Pr[A_2] + \ldots Pr[A_n] = Nq$. Thus we finally have

$$B = 1 - Pr[A] \geq 1 - Nq. \tag{4}$$

REMARK 6.1. *Note that the difference between the approximation and the lower bound (Eq. 3 and Eq. 4) is given by* $\sum_{j=2}^{N} \binom{N}{j}(-q)^j = O(q^2)$. *Thus, as* $K \to \infty$, $q \to 0$ *and the relative difference between the these expressions tends to 0.*

Figure 5 demonstrates that the lower bound is tight to the approximation in high blocking probabilities, and in fact for blocking probability higher than 80% *the delta is less than one percent. We received the same result when we repeated these checks with different N and M values, where we chose the range of the values suited to our problem. We checked also the distance of the exact solution from the approximation, but due to difficulty in precision in calculating this numerical equation of the exact solution for a large M we compared it only for a small values of M. We found out that the exact solution is between the lower bound and the approximation. From this point we will use the approximation equation as our base formula for understanding the model.*

REMARK 6.2. **(Multiple Harvesters)** *If H concurrent independent harvesters are used at each experiment then Eq. 3 changes to:* $B \approx (1-q)^N = (1-(1-N/M)^{HK})^N$.

Discussion:

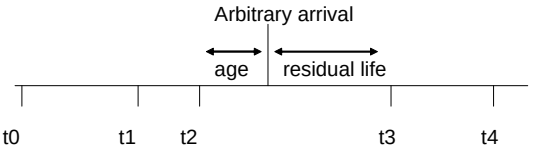


Figure 6: Illustration of the age and residual life

Assume the number of SN's is $M = 50,000$ and the number of addresses stored in a Skype client is $N = 200$. Then, using the above analysis one can easily verify that at after conducting $K = 2450$ iterations one reaches a 99% blocking probability both by the approximation and by the lower bound. In practice each iteration by a harvester takes about 2 minutes. If one uses 71 harvesters then one would reach this blocking probability after 70 minutes.

7. MODELING AND ANALYSIS OF DYNAMIC SN ADRESSES

The analysis so far is based on the assumption that SN's are permanent and have permanent IP addresses. Below we consider the approximate model proposed in Section 6.2 and extend it to dynamic addresses.

7.1 Preliminaries - The Theory of Backward and Forward Recurrence Times

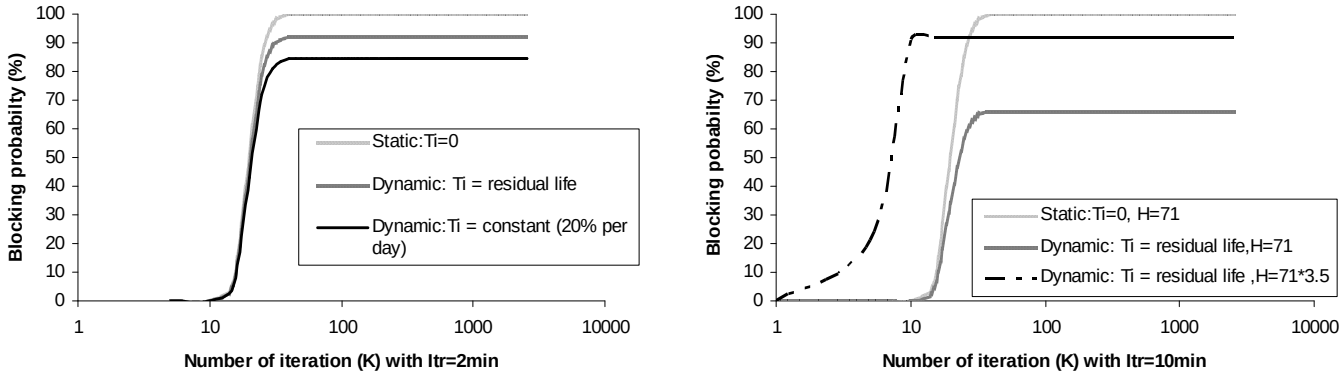
The following definition is taken from [26]: Suppose events occur at times $T_1, T_2, ...$ such that the inter-event times $T_k - T_{k-1}$ are mutually independent, positive random variables with a common cumulative distribution function. Choose an arbitrary time t. The *backward recurrence time* at t is the elapsed time since the most recent occurrence of an event prior to t. The *forward recurrence time* at t is the time from t to the next occurrence. Note that the backward and forward recurrence times are also called the *age* and *residual life* (see, e.g. [27]). See Figure 6.

Let L be a random variable denoting the durations $T_k - T_{k-1}$ with a with a common cumulative distribution function, $L(x) = Pr[L \leq x]$ and probability density function $l(x) = dL(x)/dx$. Let L_F, L_B be random variables denoting the forward recurrence time and backward recurrence time associated with L, respectively. It is well known that L_F and L_B have the same distributions whose density function obey (see. e.g.,[27])

$$l_F(x) = l_B(x) = l(x)/(1-L(x)). \tag{5}$$

7.2 Model and Analysis

We use the following to model the dynamics of IP addresses: An IP address that is given to an Internet entity will remain valid for a period of time that we call "the life of the address". Once this duration is over, the address changes and the new address will be



(a) (b)

Figure 7: (a) Blocking probability as a function of the iteration number for iteration length of 2 minutes for the different models with parameters N=50K,M=200,H=71 (b) Blocking probability as a function of the iteration number for iteration length of 10 minutes for the different models with parameters N=50K,M=200,H=71

valid for another period. We assume that the durations of all these periods are mutually independent random variables (each denoted L) with a common cumulative distribution function, $L(x) = Pr[L \leq x]$ and density function $l(x) = dL(x)/dx$. A study that provided some estimates of $l(x)$ is [25].

The processes of address harvesting and IP address dynamics interact as follows: At time $t_0 = 1$ the harvesting starts. At that epoch there are M SN addresses, all unknown to the harvesters. The harvesters then operate for a period consisting of K harvesting iterations. We normalize time to the duration of a single harvest, thus harvesting operations start at $t = 1, 2, ...$ and the SN collection iterations occur at times $t = 1, ..., K$. Address changes are assumed to occur at epochs t^+ (namely, just after the harvest starts), and assumed to be relevant to the harvest process only at t+1. For simplicity of presentation it is also convenient to assume that random variable L is given in discrete time units (and so are $L(x)$ and $l(x)$), as well as the corresponding entities for the backward recurrence time L_B.

We are now interested in deriving the number of addresses that are unknown to the harvesters at time K. To this end consider an arbitrary address a whose age (backward recurrence time) at time K is L_B. Then, a is unknown at K if and only if it was not harvested in iterations $K-L_B+1, K-L_B+2, ..., K$. Thus, similarly to the derivation of Eq. 3, the probability q that a is unknown to the harvester at K is given by

$$q = (1 - N/M)^{min\{K,L_B\}} \tag{6}$$

where $min\{K, L_B\}$ is the number of harvesting attempts applied to a after the last time it changed its address. Now, using the distribution of L_B, q is given by:

$$q = \sum_{x=1}^{\infty} l_B(x)(1 - N/M)^{min\{K,x\}}. \tag{7}$$

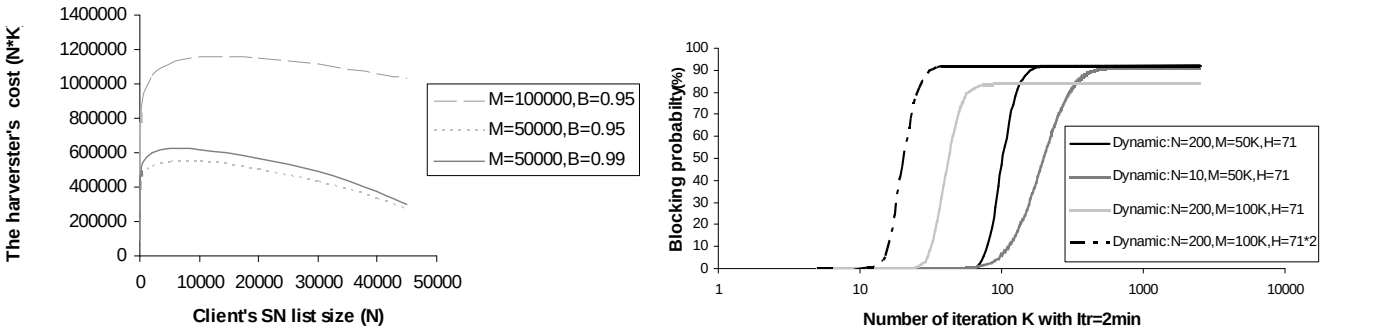
Now, following the analysis of the static addresses and our findings (see Section 6) that the formula in Eq. 3 approximates very well the exact blocking probability, we may *assume* that the blocking events of the different addresses are *mutually independent*. While this is not exact the analysis in Section 8 suggests that this a leads to a very good approximation. Thus, the blocking probability can be approximated, similarly to Eq. 3:

$$B \approx \left(1 - \sum_{x=1}^{\infty} l_B(x)(1 - N/M)^{min\{K,x\}}\right)^N. \tag{8}$$

REMARK 7.1. **(Multiple Harvesters)** *If H independent harvesters are used, then Eq. 8 changes to $B \approx \left(1 - \sum_{x=1}^{\infty} l_B(x)(1 - N/M)^{Hmin\{K,x\}}\right)^N$.*

To conclude the dynamic address analysis one needs to estimate the density function $l_B(x)$. Estimating the age of an address might be impractical (hard to measure when an address recently changes in the past). However, estimating $l_F(x)$ is much simpler since it can be done by collecting a set of addresses, and tracking them from the collection epoch until they become unavailable namely "die"). Alternatively one can attempt estimating $l(x)$ as done in [25]. One can then use Eq. 5 to compute $l_B(x)$ from $l_F(x)$ or from $l_(x)$. For our analysis, as will be demonstrated in Section 8, we choose the former.

8. HARVESTING EFFECTIVENESS: PERFORMANCE EVALUATION



(a) (b)

Figure 8: (a) The harvesting cost as a function of the client list size for analytic static model (b) Blocking probability as a function of the iteration number for the analytic dynamic model with the measured residual life for different parameters of N, M with H=71

In this section we study the effectiveness of address harvesting using the equations derived in the modeling sections.

8.1 Effect of Address Dynamics

We start by investigating the blocking probability as a function of the iteration number for three of the following models, assuming M=50K, N=200, H=71 (Figure 7 (a)). First, we consider a static address model, depicted in the upper curve. It is clear that in the static model - the blocking probability can reach the maximum, 100%, quickly, just after 50 iterations. Note that in the static model, where the population is fixed, one can always reach the 100% blocking probability - regardless of the number of harvesters. The number of harvesters only influences the time it takes to reach 100% blocking. In the dynamic models, in contrast, there is a clear upper bound on the maximal blocking probability one can achieve with specific number of harvesters. This is due to the fact that the population is changing with time, hence if there are no enough parallel harvesters one cannot learn short life span SNs.

Second, we consider a dynamic address model where the SN residual life is identical to what we recorded in the experimental measurements shown in Figure 4 (18%, 8% , 5% ,3% die in days 1, 2, ,3, 4 ...); Thus it represents a "realistic situation". This is depicted in the middle curve. For this setup the blocking rate achieved is 92%, close to our experimental result of 95%. Third, we examine how the address dynamics rate affects performance, and consider a case where the residual life is distributed as 20% on each of the days 1, 2, 3, 4, 5 (uniform over the day), representing much faster address churn than in the second (realistic case); this is depicted in the lower curve. One can see that the blocking probability is quite sensitive to the address dynamics rate, reaching only 82% in this case.

8.2 Effect of Skype's "slow down"

In order to reduce its vulnerability to address harvesting, the Skype system can adopt a strategy of releasing addresses to the clients in a "slow mode". This can be very effective when the addresses are dynamic (harvesters do not have enough time to harvest the address). In Figure 7 (b) we depict the blocking probability in a case where the iteration time is 10 minutes (i.e., the harvester receive new SN list every 10 minutes) as opposed to 2 minutes iteration time in Figure 7 (a). The impact on the maximal blocking probability is significant only for the case of dynamic addresses. One can observe it in the solid-dark curve, depicting the blocking under the experimentally-measured residual life distribution (from Figure 4) and reaching only 60% blocking. However, as can be seen in the dashed curve, one can overcome this Skype's defence by increasing the number of harvesters. Specifically, if the iteration duration is set to be L times longer, then one needs less than L times more harvesters. In the figure we show that if we use $71 \cdot 3.5$ harvesters we can reach the same blocking probability as with the case of 2 minute duration iterations (i.e.m blocking probability of 92%).

8.3 Effect of List Size and Population Size

One might wonder whether Skype can reduce the vulnerability to harvesting by modifying the client list size N. In the static model, if we take the approximation Equation 3 and find K (the number of harvesting iterations needed) as a function of B (the desired blocking probability) we find that $K = log_{(1-\frac{N}{M})}(1 - B^{\frac{1}{N}})$. It

easy to see that K is inversely proportional to N. I.e., reducing N will increase the number of required iterations by the harvester. However, the harvesting cost is not dominated by the number of required iterations, but by the number of required records it needs to collect. Hence, we are more interested in KN which is equal to $log_{(1-\frac{N}{M})}(1-B^{\frac{1}{N}}) \cdot N$ which results with a more complex relationship between N and the harvester cost. On the one hand when the list size is large, the harvester enjoys the fact that it discovers a high number of unique SNs in each iteration. However, on the other hand, the larger the client's list, the more SNs the harvester needs to discover, since it needs to discover all the SNs in the client list in order to block the client.

Figure 8 (a) shows the harvesters cost as a function of the list size for different population sizes, M, and different required blocking probability B for the static model. We can learn from the figure that by using the optimal list size (optimal from Skype's perspective), Skype can increase the cost of the harvesters only by less than 25% (from 450K to around 540K, for $B = 95\%$ and $M = 50K$). Hence, changing the list size, is not an effective remedy against SN harvesting.

Figure 8(b) shows the blocking probability for the dynamic model with the experimentally measured SN residual life for different parameters. It is clear that reducing N, the list size, has again a minor impact on the maximal blocking probability. It only increases the time needed in order to reach the maximal blocking probability. Increasing the population size, M, has more impact on the achievable maximal blocking probability, and this is true since it influences the number of short life span SNs that the harvesters need to learn in short time. Again, by increasing the number of harvesters, we can reach the same blocking probability. In this case if the population size increase by a factor of two, we need to increase the number of harvesters by a factor of two as well.

9. CONCLUSION AND FUTURE WORK

In this paper we use measurements and modeling and show that it is feasible to harvest the vast majority of the active SNs of Skype. This makes Skype vulnerable to filtering according to SN lists. We suspect that this vulnerability is not unique to Skype, and may be a fundamental problem of $p2p$ networks whose topology can be discovered and used to block the protocol despite their distributed nature.

The Skype network is a Super Node based system. In future work we plan to investigate the use of the harvesting technique to block the two other types of P2P networks: 1) Fully distributed systems, like the original Gnutella or BubbleStorm and 2) P2P Systems based on distributed hash table, like Chord and Can.

Acknowledgments: We would like to thank Sheldon Ross for suggesting the approach presented in Equation 9 and Michael Tarsi for helpful suggestions. We would like to thank Jussi Kangasharju for helpful discussions and suggestions for future work.

10. REFERENCES

[1] ichael Gough, S. A. Baset, J. Brashars, L. Chaffin, M. Cross, D. Doug, and M. Sweeney, *Skype Me*. Syngress Publishing, first ed., 2006.
[2] "kazaa. [Online] 2007." http://kazaa.com.
[3] "Skype. Take a deep breath." http://Skype.com.
[4] S. A. Baset and H. G. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," in *Proc. INFOCOM 2006*, 2006.
[5] K. F. Suh, D. R. Kurose, and D. J. Towsley, "Characterizing and detecting skype-relayed traffic," in *Proc. INFOCOM 2006*, 2006.
[6] S. Ehlert, T. Magedanz, S. Petgang, and D. Sisalem, "Analysis and signature skype voip session traffic," Tech. Rep. NGNI-SKYPE-06B, 2006.
[7] "D. Fabrice. Skype uncovered, 2005.." http://www.ossir.org/windows/supports/listewindows-2005.shtml.
[8] "Recon 2006 - Fabrice Desclaux and Kostya Kortchinsky - Vanilla Skype (2006)." http://recon.cx/en/f/vskype-part1.pdf , http://recon.cx/en/f/vskype-part2.pdf.
[9] "Phillipe Biondi and Fabrice Desclaux. Silver needle in the Skype. Presentation at BlackHat Europe, March 2006." www.blackhat.com/html/bh-media-archives/bh-archives-2006.html.
[10] S. Guha, N. Daswani, and R. Jain, "An experimental study of the skype peer-to-peer voip system," in *IPTPS06, Feb. 2006.*, 2006.
[11] L. D. Cicco, S. Mascolo, and V. Palmisano, "An experimental investigation of the congestion control used by skype voip," in *5th International Conference on Wired/Wireless Internet Communications,May 2007*, 2007.
[12] "Apoc Matrix , The SkypeLogger, December 2006." http://www.epokh.org/drupy/files/Skype%20Logger.pdf.
[13] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei., "Quantifying skype user satisfaction," in *SIGCOMM, 2006*, 2006.
[14] T. Hofeld, A. Binzenhofer, M. Fiedler, and K. Tutschku, "Measurement and analysis of skype voip traffic in 3g umts systems," in *The 4th InternationalWorkshop on Internet Performance, Simulation, Monitoring and Measurement (IPS-MoMe 2006)*, 2006.
[15] X. Wang, S. Chen, and S. Jajodia, "Tracking

anonymous peer-to-peer voip calls on the internet," in *ACM Conference on Computer and Communications Security (November 2005)*, 2005.
[16] "Skype. SkypeOut. Skype. [Online] 2007." http://www.Skype.com/products/skypeout/.
[17] "Skype SMS. Skype. [Online] 2007." http://www.Skype.com/products/skypesms/.
[18] "Active and run two (or more) Skype profiles at the same time." http://www.henshall.com/docs/ipod%20RadioonSkypeJan2005.pdf.
[19] "SPam and Open Relay Blocking system (SORBS)." http://www.au.sorbs.net/.
[20] S. S. and S. Gummadi, P. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Multimedia Computing and Networking. 2002, MMCN'02, San Jose, CA, Jan*, 2002.
[21] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *ACM SIGCOMM Internet Measurement Workshop, Nov. 2002*, 2002.
[22] R. Bhagwan, S. Savage, , and G. Voelker., "Understanding availability," in *IPTPS 03, 2003*, 2003.
[23] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, , and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *ACM SOSP, 2003*, 2003.
[24] P. B. Godfrey, S. Shenker, , and I. Stoica, "Minimizing churn in distributed systems," in *ACM SIGCOMM 2006*, 2006.
[25] Y. Xie, K. A. F. Yu, E. Gillum, M. Goldszmidt, and T. Wobber, "How dynamic are ip addresses," in *ACM SIGCOMM, 2007*, 2007.
[26] S. I. Gass and C. M. Harris, *Encyclopedia of Operations research and Management Science, 2nd Edition*. Kluwer Academic Publishers, second ed., 2001.
[27] L. Kleinrock, *Queueing Systems, Volume I: Theory*. John Wiley and Sons, first ed., 1975.
[28] R. S. Robinson, N. Hudson, D. Walker, and L. Gilman, "Copuon collector report."

APPENDIX

An alternative approach for computing $X(l,k)$ is as follows:

$$X(l,k) = \binom{M}{l} \left[\frac{\binom{l}{N}}{\binom{M}{N}}\right]^k \sum_{j=0}^{l} (-1)^j \binom{l}{j} \left[\frac{\binom{l-j}{N}}{\binom{l}{N}}\right]^k. \quad (9)$$

The equation can be derived from the literature on the "baseball card collector problem". The basic baseball card collector problem is: Given a set of M cards and a collector who receives k packs each consisting of N cards. What is the probability that the collector collects all M cards. Using our notation above, this should be given by $X(M,k)$.

$X(l,k)$ can be computed by fixing a SET of l cards and computing the product of: (i) Pr[the cards collected are all from SET] and (ii) Pr [all card types are collected | the overall set of cards is SET]. Then the result can be obtained multiplying the product by: (iii) the number of different ways SET can be chosen.

(i) is given by $\left[\frac{\binom{l}{N}}{\binom{M}{N}}\right]^k$. (ii) reflects the probability that the collector collects all types when the number of cards is l. This is given (see [28]) by $\sum_{j=0}^{l}(-1)^j\binom{l}{j}\left[\frac{\binom{l-j}{N}}{\binom{l}{N}}\right]^k$. Lastly (iii) is simply $\binom{M}{l}$.

The sum in Eq. 9 can be approximated (see [28]) by $e^{-e^{-c}}$ where $c = Nk/l - logl$.

A special case of Eq. 9 is the case where $N = 1$. IN this case one gets the coupon collector problem. In this case $X(M,k)$ (getting all M cards in k attempts) is given by : $X(M,k) = \sum_{j=0}^{M}(-1)^j\binom{M}{j}(1-j/M)^k$.