ENSC 835: Communication Networks
Spring 2011

Final Project
Internet Endpoints Profiling And Trace Analysis

http://www.sfu.ca/~smmohamm/current_projects/default.html

Seyed Mojtaba Mohammadian Abkenar
Student ID : 301138241
mojtaba_abkenar@sfu.ca

Abstract

Profiling is emerging as a useful tool for a variety of diagnosis and security applications. At a global scale we can consider Internet and social networks as the networks of interests for Profiling.

Classifying traffic flows according to the applications that generate them is an important task for (a) effective network planning and design, and (b) monitoring the trends of the applications in operational networks. Political, economic, and legal struggles over appropriate use and pricing of the Internet have brought the issue of traffic classification to mainstream media. In all cases the algorithmic playing field is traffic classification: stopping or deprioritizing traffic of a certain type, versus obfuscating a traffic profile to avoid being thus classified. Traffic classification is also relevant to the more mundane but no less important task of optimizing current network operations and planning improvements in future network architectures, which means the increasing incentives to prevent accurate classification of one's own traffic presents an obstacle to understanding, designing, operating, financing, and regulating the Internet.

Existing profiles are often narrowly focused in terms of the data they capture or the application they target. Understanding Internet access trends at a global scale, i.e., how people use the Internet, is a challenging problem that is typically addressed by analyzing network traces. In this project, I study different methods, difficulties and available tools for profiling Internet. As part of the project, I write an application for creating profile based "Googling the Internet: Profiling Internet Endpoints via the World Wide Web" paper.

Table of Contents:

1. Introduction

Understanding what people are doing on the Internet at a global scale, e.g., which applications and protocols they use, which sites they access, and who they try to talk to, is an intriguing and important question for a number of reasons. Answering this question can help reveal fascinating cultural differences among nations and world regions. It can shed more light on important social tendencies and help address imminent security vulnerabilities. Moreover, understanding shifts in clients' interests, e.g., detecting when a new application or service becomes popular, can dramatically impact traffic engineering requirements as well as marketing and IT-business arenas.

In the early Internet, traffic classification relied on the use of transport layer port numbers, typically registered with IANA to represent a well-known application. More recently, increasingly popular applications such as those that support peer-to-peer (P2P) file sharing hide their identity by assigning ports dynamically and/or using well-known ports of other applications, rendering port-based classification less reliable. A more reliable approach adopted by commercial tools inspects packet payloads for specific string patterns of known applications. While this approach is more accurate, it is resource-intensive, expensive, scales poorly to high bandwidths, does not work on encrypted traffic, and causes tremendous privacy and legal concerns. Three proposed traffic classification approaches that avoid payload inspection are: (1) host-behavior-based, which takes advantage of information regarding "social interaction" of hosts , (2) flow features-based, which classifies based on flow duration, number and size of packets per flow, and inter-packet arrival time , and (3) search engine-based.

Despite many proposed algorithms for traffic classification, there are still no definitive answers to pragmatic questions: What is the best available traffic classification approach? Under what link characteristics and traffic conditions does it perform well, and why? What are the fundamental contributions and limitations of each approach?

There are four traffic classification approaches: port-based, host-behavior, flow-features-based, and search-engine-based. In this project, I am trying to introduce these methods and analyze the advantages and limitations of each approach, evaluate methods to overcome the limitations, and extract insights and recommendations for both the study and practical application of traffic classification.

**I will use a lot of jargons and acronyms related to software engineering in this report, curious reader can follow the hyperlink for more information.

2. Blind classification (BLINC)

BLINC is an approach to classifying traffic flows according to the applications that generate them. This approach is based on observing and identifying patterns of host behavior at the transport layer. It analyzes these patterns at three levels of increasing detail (i) the social, (ii) the functional and (iii) the application level. Classifying traffic flows according to the applications that generate them is an important task for (a) effective network planning and design, and (b) monitoring the trends of the applications in operational networks. BLINC addresses the traffic classification problem with the following constraints: (i) no access to user payload is possible, (ii) well-known port numbers cannot be assumed to indicate the application reliably, and (iii) we can only use the information that current flow collectors provide.

Each level of classification provides increasing knowledge of host behavior, identifying specific applications depends on the unveiled "cross-level" characteristics.

- At the social level, BLINC captures the behavior of a host as indicated by its interactions with other hosts. First, it examines the popularity of a host. Second, it identifies communities of nodes, which may correspond to clients with similar interests or members of a collaborative application.
- At the functional level, BLINC captures the behavior of the host in terms of its functional role in the network, namely whether it acts as a provider or consumer of a service, or both, in case of a collaborative application. For example, hosts that use a single port for the majority of their interactions with other hosts are likely to be providers of the service offered on that port.
- At the application level, BLINC captures the transport layer interactions between particular hosts on specific ports with the intent to identify the application of origin. First, it provides a classification using only 4-tuples (source address, destination address, source port, and destination port). Then, it refines the classification further by exploiting other flow characteristics such as the transport protocol or the average packet size.

A key feature of this methodology is that it provides results at various levels of detail and accuracy. First, BLINC analyzes traffic at the aforementioned three levels. Second, the classification criteria are controlled by thresholds that when relaxed or tightened, achieve the desired balance between an aggressive and a conservative classification. The level of accuracy and detail may be chosen according to: (a) the goal of the study, and (b) the amount of exogenous information (e.g., application specifications).

BLINC provides two types of output. First, it reports aggregate per-class statistics, such as the total number of packets, flows and bytes. Second, it produces a list of all flows (5-tuple) tagged with the corresponding application for every time interval. Furthermore, BLINC can detect unknown or non-conformant hosts and flows.

2.1. Classification at the social level

BLINC identifies the social role of each host in two ways. First, it focuses on its popularity, namely the number of distinct hosts it communicates with. Second, it detects communities of hosts by identifying and grouping hosts that interact with the same set of hosts. A community may signify a set of hosts that participate in a collaborative application, or offer a service to the same set of hosts.

Examining the social behavior of single hosts. The social behavior of a host refers to the number of hosts this particular host communicates with. To examine variations in host social behavior, Fig. 1 presents the complementary cumulative distribution function (CCDF) of the host popularity. Based on payload classification from section 3, we display four different CCDFs corresponding to different types of traffic, namely web, p2p, malware (e.g., failed no payload connections on known malware ports), and mail. In all cases, the majority of sources appear to communicate with a small set of destination IPs.

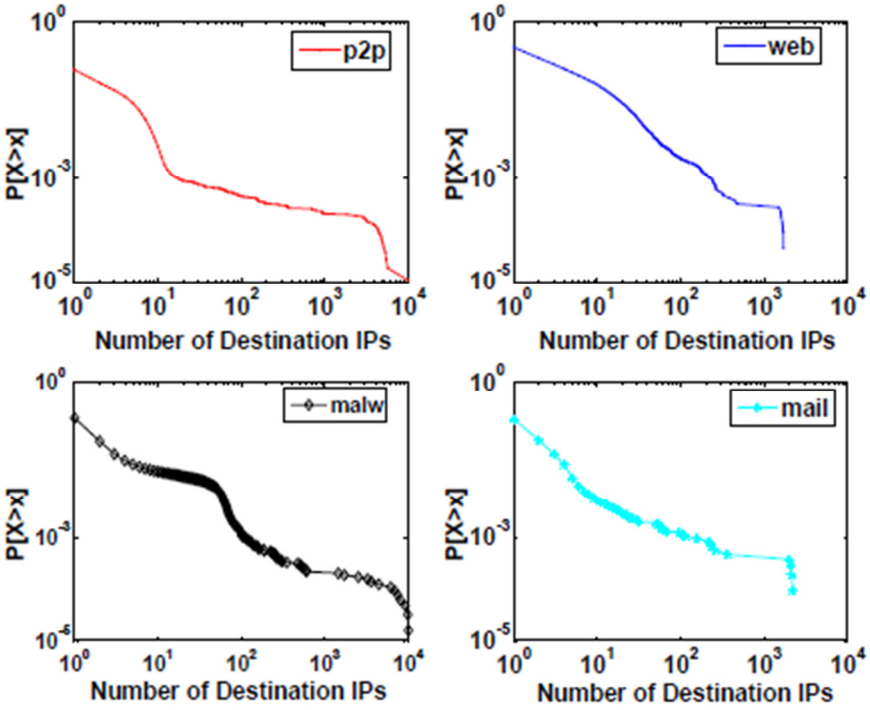

Figure 1: Complementary cumulative distribution function of destination IP addresses per source IP for 15 minutes of the UN1 trace for four different applications.

In general, the distribution of the host popularity cannot reveal specific rules in order to discriminate specific applications, since it is highly dependent upon the type of network, link or even the specific IPs. However, this distribution allows us to distinguish significant differences among applications.

Detecting communities of hosts. Social behavior of hosts is also expressed through the formation of communities or clusters between sets of IP addresses. Communities will appear as bipartite cliques in the traces, like the one shown in Fig. 2. The bipartite graph is a consequence of the single observation point. Interactions between hosts from the same side of the link are not visible, since they do not cross the monitored link. Communities in bipartite graph can be

either exact cliques where a set of source IPs communicates with the exact same set of destination IPs, or approximate cliques where a number of the links that would appear in a perfect clique is missing. Communities of interest have also been studied, where a community is defined by either the popularity of a host or frequency of communications between hosts. Our definition of community is targeted to groups of hosts per application type.

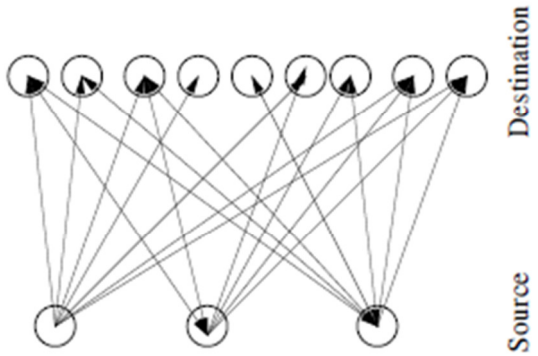

Figure 2: An example of a community in the traces: the graph appears as an approximate bipartite clique.

Identifying the communities is far from trivial, since identifying maximal cliques in bipartite graphs is an NP-Complete problem. However, there exist polynomial algorithms for identifying the cross-associations in the data mining context. Cross-association is a joint decomposition of a binary matrix into disjoint row and column groups such that the rectangular intersections of groups are homogeneous or formally approximate a bipartite clique. In our case, this binary matrix corresponds to the interaction matrix between the source and destination IP addresses in the traces.

In general, three different types of communities can be found, according to their deviation from a perfect clique:

"Perfect" cliques: a hint for malicious flows. While the previous example displays a perfect clique in gaming traffic, we find that perfect cliques are mostly signs of malicious behavior.

Partial overlap: collaborative communities or common interest groups. In numerous cases, only a moderate number of common IP addresses appear in the destination lists for different source IPs. These cases correspond to peer-to-peer sources, gaming and also clients that appear to connect to the same services at the same time (e.g., browsing the same web pages, or streaming).

Partial overlap within the same domain: service "farms". Closer investigation of partial overlap revealed hosts interacting with a number of IP addresses within the same domain, e.g., IP addresses that differ only in the least significant bits. Payload analysis of these IPs revealed that this behavior is consistent with service "farms": multi-machine servers that load-balance requests of a host to servers within the same domain. We find that service "farms" were used to offer web, mail, streaming, or even dns services.

Conclusion and Rules: Based on the above analysis, we can infer the following regarding the social behavior of network hosts. First, "neighboring" IPs may offer the same service (e.g., server farms). Thus, identifying a server might be sufficient to classify such "neighboring" IPs

under the same service (if they use the same service port). Second, exact communities may indicate attacks. Third, partial communities may signify p2p or gaming applications. Finally, most IPs act as clients having a minimum number of destination IPs. Thus, focusing on the identification of the small number of servers can retrospectively pinpoint the clients, and lead to the classification of a large portion of the traffic, while limiting the amount of associated overhead. Identification of server hosts is accomplished through the analysis of the functional role of the various hosts.

2.2. Classification at the functional level

At this level, BLINC identifies the functional role of a host: hosts may primarily offer services, use services, or both. Most applications operate with the server-client paradigm.

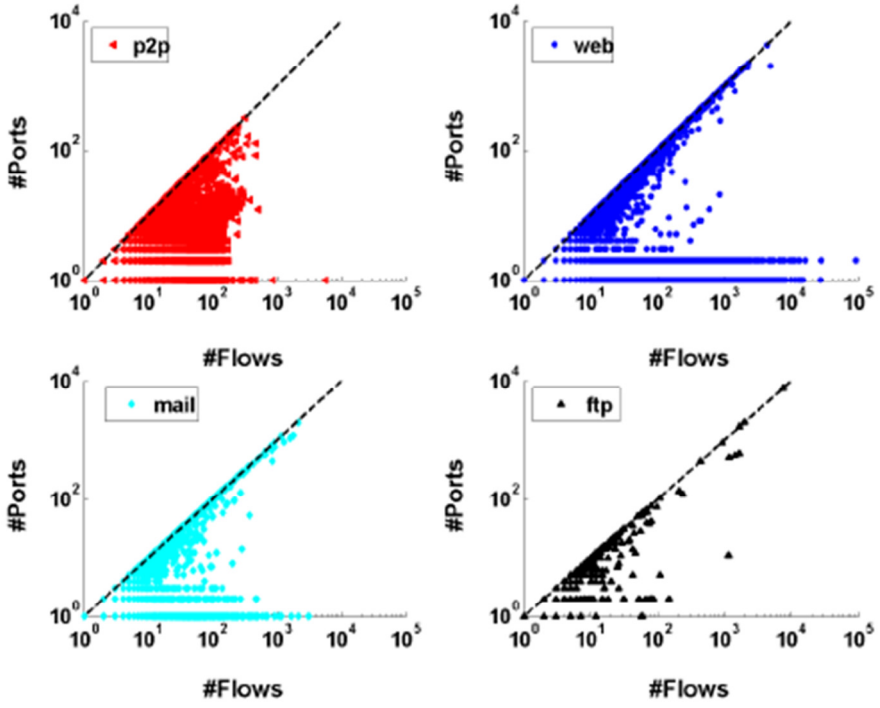

Figure 3: Number of source ports versus number of flows per source IP address in the UN1 trace for a 15-minute interval for four different applications. In client-server applications (web,ftp,mail), most points fall on the diagonal or on horizontal lines for small values in the y-axis (number of used ports). In p2p, points are clustered in-between the diagonal and the x-axis.

Each subplot of Fig. 3 presents traffic from a different application as identified by payload analysis. It identifies three distinct behaviors:

Typical client-server behavior: Client-server behavior is most evident for web traffic (Fig. 3, top-right), where most points fall either on the diagonal or on horizontal lines parallel to the x-axis for small values of y (less or equal to two). The first case (where the number of ports is equal to the number of distinct flows) represents clients that connect to web servers using as

many ephemeral source ports as the connections they establish. The latter case reflects the actual servers that use one (y = 1, port 80, HTTP) or two (y = 2, port 80, HTTP and 443, HTTPS) source ports for all of their flows.

Typical collaborative behavior: In this case, points are clustered between the x-axis and the diagonal, as shown in the p2p case in Fig. 3 (top-left), where discrimination between client and server hosts is not possible.

Obscure client-server behavior: In Fig. 3, we plot the behavior for the case of mail and ftp. While mail and ftp fall under the client-server paradigm, the behavior is not as clear as in the web case for two reasons:

- The existence of multiple application protocols supporting a particular application, such as mail. Mail is supported by a number of application protocols, i.e., SMTP, POP, IMAP, IMAP over SSL, etc., each of which uses a different service port number. Furthermore, mail servers often connect to Razor databases through SpamAssassin to report spam. This practice generates a vast number of small flows destined to Razor servers, where the source port is ephemeral and the destination port reflects the SpamAssassin service. As a result, mail servers may use a large number of different source ports.
- Applications supporting control and data streams, such as ftp. Discriminating client-server behavior is further complicated in cases of separate control and data streams. For example, passive ftp, where the ftp server uses as source ports a large number of ephemeral ports different than the service ports (20,21), will conceal the ftp server.

Conclusion and Rules: If a host uses a small number of source ports, typically less or equal to two, for every flow, then this host is likely providing a service. Our measurements suggest that if a host uses only one source port number, then this host reflects a web, a chat or a SpamAssassin server in case of TCP, or falls under the Network Management category in case of UDP.

2.3. Classification at the application level

In this level, I combine knowledge from the two previous levels coupled with transport layer interactions between hosts in order to identify the application of origin. The basic insight exploited by this methodology is that interactions between network hosts display diverse patterns across the various application types. I first provide a classification using only the 4-tuple (IP addresses and ports), and then, I refine it using further information regarding a specific flow, such as the the protocol or the average packet size.

I model each application by capturing its interactions through empirically derived signatures. I visually capture these signatures using graphlets that reflect the “most common” behavior for a particular application. A sample of application-specific graphlets is presented in Fig. 4. Each graphlet captures the relationship between the use of source and destination ports, the relative cardinality of the sets of unique destination ports and IPs as well as the magnitude of these sets.

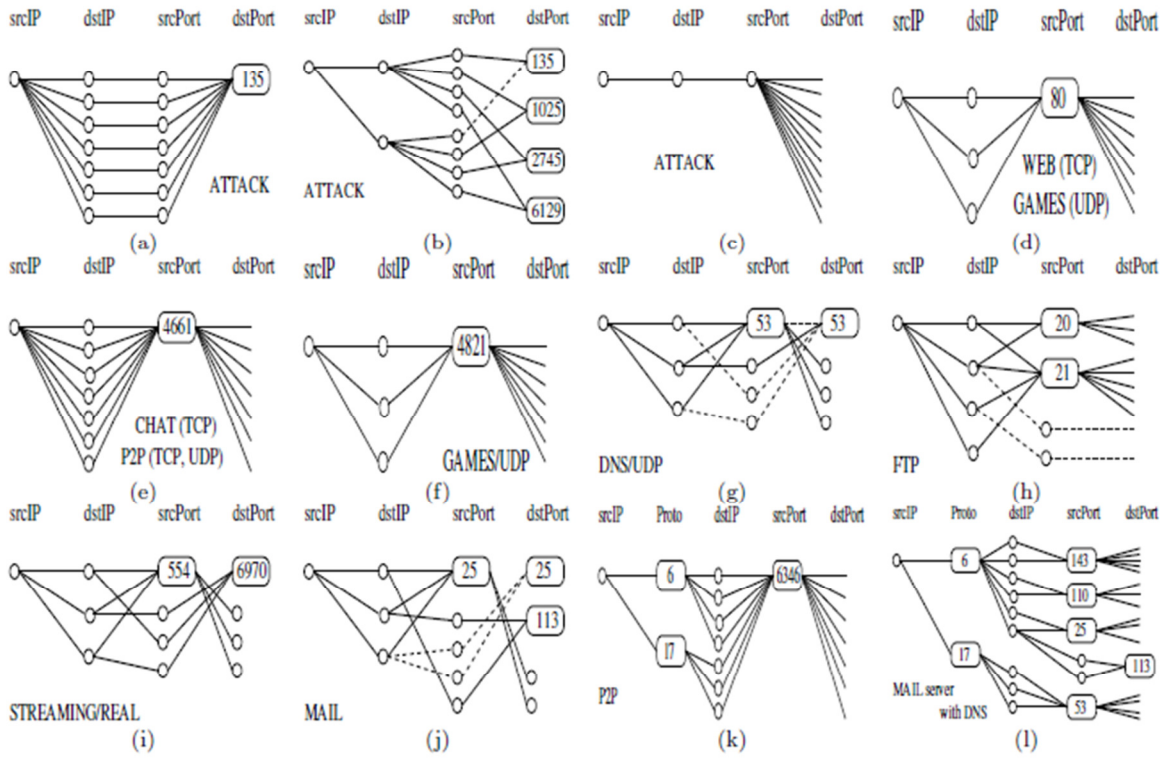

Figure 4: Visual representation of transport-layer interactions for various applications: port numbers are provided for completeness but are not used in the classification.

Having a library of these graphlets, allows us to classify a host by identifying the closest matching behavior. Since unknown behavior may match several graphlets, the success of the classification will then have to rely on operator-defined thresholds to control the strictness of the match.

In more detail, each graphlet has four columns corresponding to the 4-tuple source IP, destination IP, source port and destination port. I also show some graphlets with 5 columns, where the second column corresponds to the transport protocol (TCP or UDP) of the flow. Each node2 presents a distinct entry to the set represented by the corresponding column, e.g., 135 in graphlet 5(a) is an entry in the set of destination ports. The lines connecting nodes imply that there exists at least one flow whose packets contain the specific nodes (field values). Dashed lines indicate links that may or may not exist and are not crucial to the identification of the specific application. Note that while some of the graphlets display port numbers, the classification and the formation of graphlets do not associate in any way a specific port number with an application.

The order of the columns in our visual representation of each graphlet mirrors the steps of our multilevel approach. Our starting field, the source IP address, focuses on the behavior of a particular host. Its social behavior is captured in the fan out of the second column which corresponds to all destinations IPs this particular source IP communicates with. The functional role is portrayed by the set of source port numbers. For example, if there is a “knot” at this level the source IP is likely to be a server as mentioned before. Finally, application types are distinguished using the relationship of all four different fields. Capturing application specific

interactions in this manner can distinguish diverse behaviors in a rather straightforward and intuitive manner as shown in Fig. 4.

The power of this method lies in the fact that we do not need to know the particular port number ahead of time. The surprising number of flows at the specific port will raise the suspicion of the network operator.

3. Search engine-based classification

The key hypothesis of this approach is that most of the information needed to profile the Internet endpoints is already available around us—on the Web. Web-based "unconstrained endpoint profiling" (UEP) approach shows advances in the following scenarios:

- Even when no packet traces are available, it can accurately infer application and protocol usage trends at arbitrary networks.
- When network traces are available, it outperforms state-of-the-art classification tools such as BLINC.
- When sampled flow-level traces are available, it retains high classification capabilities.

The goal is to characterize endpoints by strategically combining information available at a number of different sources on the Web. The key is that records about many Internet endpoints' activities inevitably stay publicly archived. Of course, not all active endpoints appear on the Web, and not all communication leaves a public trace. Still, enormous amounts of information does stay publicly available, and that a 'purified' version of it could be used in a number of contexts

3.1 Unconstrained Endpoint Profiling

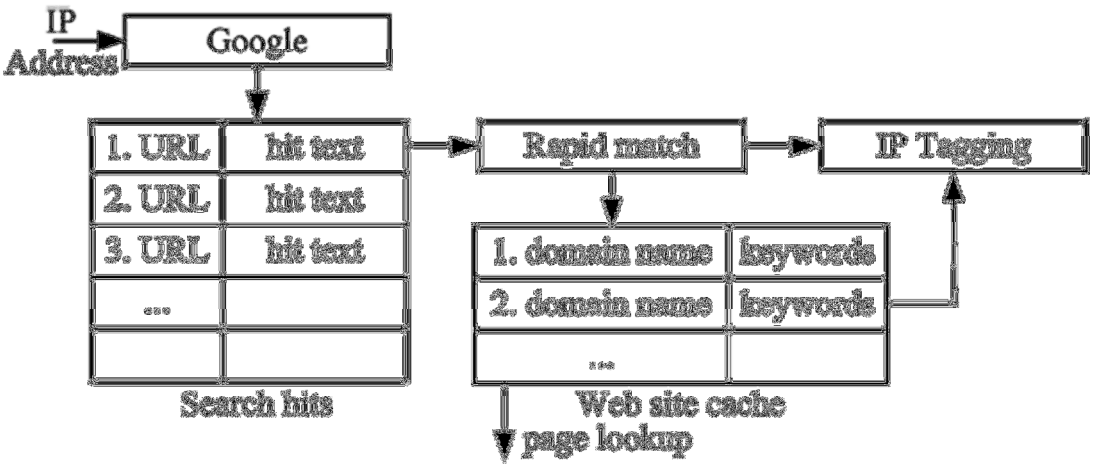

Fig. 5. Web-based endpoint profiling tool. Generates IP address tags based on information found via Google.

Fig. 5 depicts the search engine-based endpoint profiling tool. At the functional level, the goal is straightforward: we query the search engine by searching on text strings corresponding to the standard dotted decimal representation of IP addresses. We collect search hits returned by search engine, and then extract information about the corresponding endpoint. The output is a set of *tags* (features) associated with this IP address. Such information can come from a number of different URLs.

TABLE I
KEYWORDS—WEB SITE CLASS—TAGS MAPPING

Keywords	Website Class	Tags
{'ftp' | 'webmail' | 'dns' | 'email' | 'proxy' | 'smtp' | 'mysql' | 'pop3' | 'mms' | 'netbios'}	Protocols and Services	<protocol name> server
{'trojan' | 'worm' | 'malware' | 'spyware' | 'bot'}	Malicious information list	<issue name> affected host
'spam'	Spamlist	spammer
{'blacklist' | 'banlist' | 'ban' | 'blocklist'}	Blacklist	blacklisted
'adserver'	Ad-server list	adserver
{'domain' | 'whois' | 'website'}	Domain database	website
{'dns' | 'server' | 'ns'}	DNS list	DNS server
{'proxy' | 'anonymous' | 'anonymizer'}	Proxy list	proxy server
'router'	Router admin log list	router
'mail server'	Mail server list	mail server
'mail server' && {'spam' | 'dictionary attacker'}	Malicious mail servers list	mail server [spammer] [dictionary attacker]
{'counter strike' | 'warcraft' | 'age of the empires' | 'quake' | 'halo' | 'game'}	Gaming servers list	<game name> server
{'counter strike' | 'warcraft' | 'age of the empires' | 'quake' 'halo' | 'game'} && {'abuse' | 'block'}	Gaming abuse list	<game> node [abuser] [blocked]
{'torrent' | 'emule' | 'kazaa' | 'edonkey' | 'announce' | 'tracker' | 'seeder' | 'limewire' | 'bittorrent' | 'azureus' | 'spitfire' | 'gnutella' }	p2p node list	<protocol name> p2p node
{'irc' | 'undernet' | 'innernet' | 'dal.net'}	IRC servers list	IRC server
{'yahoo' | 'gtalk' | 'msn' | 'qq' | 'icq' | 'server' | 'block'}	Chat servers	<protocol name> chat server
{'generated by' | 'awstats' | 'wwwstat' | 'counter' | 'stats'}	Web log site	web user [operating system] [browser][date]
{'redirect' | 'location'}	Proxy log	proxy user [site accessed]
{'forum' | 'answer' | 'response' | 'reponse' | 'comment' | 'comentario' | 'comentarios' | 'posted' | 'poste' | 'registered' | 'registrado' | 'enregistre' | 'created' | 'criado' 'cree' | 'bbs' | 'board' | 'club' | 'guestbook' | 'cafe' }	Forum	forum user [role][user name] [http share][ftp_share] [streaming node]

At a high level, our approach is based on searching for information related to IP addresses on the Web. The larger the number of search hits returned for a queried IP address, and the larger number of them confirming a given behavior (i.e., a), the larger the confidence about the given endpoint activity. The profiling methodology involves the following three modules:

1) **Rule Generation:** The process starts by querying search using a sample "seed set" of random IP addresses then obtaining the set of search hits. Each search hit consists of a URL and corresponding hit text. We then extract all the words and biwords (word pairs) from the hit texts of all the hits returned for this seed set. After ranking all the words and biwords by the number of hits they occur in and after filtering the trivial keywords (e.g., "the"), we constrain ourselves to the top N keywords that could be meaningfully used for endpoint classification. By meaningfully used we mean that the keyword found implies an application or application class associated with network activity.
   Then, in the only manual step in our methodology, we construct a set of rules that map keywords to an interpretation for the functioning of that Web site, i.e., the *Web site class*. The rules are as shown in the relationship between Column 1 and 2 in Table I.

2) **Web Classifier***:* Extracting information about endpoints from the Web is a nontrivial problem. In this approach, we first characterize a given Web page (returned by search engine), i.e., determine what information is contained on a Web site.
   Rapid URL Search. Some Web sites can be quickly classified by the keywords present in their domain name itself. Hence, after obtaining a search hit we first scan the URL string to identify the presence of one of the keywords from our keyword set in the URL and

then determine the Web site's class on the basis of the rules in Table I. Typically, Web sites that get classified by this rapid URL search belong to the Forum and Web log classes. If the Rapid URL search succeeds, we proceed to the IP tagging phase. If rapid match fails, we initiate a more thorough search in the hit text, as we explain next.

Hit Text Search. To facilitate efficient Web page characterization and endpoint tagging, we build a Web site cache. The key idea is to speed-up the classification of endpoints coming from the same Web sites/domains under the assumption that URLs from the same domain contain similar content. In particular, we implement the Web site cache as a hash table indexed by the domain part of the URL. Hence, all IPs that return a search hit from this domain can be classified in the same way. Whenever we find a URL whose corresponding domain name is not present in the cache, we update the cache as follows. First, we insert the domain name for the URL as an index into the cache with an empty list (no keywords) for the value. In addition, we insert a counter for number of queried IP addresses that return this URL as a hit along with the corresponding IP address. High values for the counter would indicate that this domain contains information useful for classifying endpoints. Thus, when the counter for number of IP addresses goes over a threshold we retrieve the Web page based on the last URL. We have chosen this threshold in order to filter out Web sites that carry information about a single IP address only. At the same time, this approach maximizes the amount of traffic that we can classify while filtering out the above sites. Then, we search the Web page for the keywords from the keyword set and extract the ones that can be found. Next, we use the rule-based approach to determine the class to which this Web site (and hence the domain) belongs. Finally, we insert an entry in the cache with the domain name as the key and the list of all associated keywords (from Table I) as the value. When a URL's domain name is found in the cache, then we can quickly classify that URL by using the list of keywords present in the cache. In this way, the cache avoids having to classify the URL on every hit and simplifies the IP-tagging phase, as we explain next.

3) *IP Tagging:* The final step is to tag an IP address based on the collected information. We distinguish between three different scenarios.

URL based tagging. In some scenarios, an IP address can be directly tagged when the URL can be classified via rapid search for keywords in the URL itself. In such scenarios, we can quickly generate the appropriate tags by examining the URL itself. In particular, we use the mapping between a Web site class (Column 2) and IP tags (Column 3) in Table I to generate the tags. In majority of the cases, such rapid tagging is not possible and hence we have to examine the hit text for additional information.

General hit text based tagging. For most of the Web sites, we are able to accurately tag endpoints using a keyword based approach. The procedure is as follows. If we get a match in the Web site cache (for the specific URL we are currently trying to match), we check if any of the keywords associated with that domain match in the search hit text. Surprisingly, we typically find at least a *single* keyword that clearly reveals the given IP's nature and enables tagging. Table I provides the mapping between the domain class and IP tags.

Hit text based tagging for Forums. The keyword-based approach fails when a URL maps

to an Internet forum site. This is because a number of non correlated keywords may appear at a forum page. Likewise, forums are specific because an IP address can appear at such a site for different reasons. Either it has been automatically recorded by a forum post, or because a forum user deliberately posted a link (containing the given IP address) for various reasons. In the case of forums, we proceed as follows. First, we use a postdate and username in the vicinity of the IP address to determine if the IP address was logged automatically by a forum post. Hence, we tag it as the forum user. If this is not the case, the presence of the following keywords: http:\ , ftp:\ , ppstream:\ , mms: \, etc. in front of the IP address string in the hit text suggests that the user deliberately posted a link to a shared resource on the forum. Consequently, we tag the IP address as an http share or ftp share, or as a streaming node supporting a given protocol (ppstream, mms, tvants, sop, etc.). Because each IP address generates several search hits, multiple tags can be generated for an IP address. Thus aggregating all the tags corresponding to an IP address either reveals additional behavior or reaffirms the same behavior. For the first case, consider the scenario where an IP address hosts multiple services that would then be identified and classified differently and thereby generate different tags for that IP address, revealing the multiple facets of the IP address' behavior. In the second case, if an IP address' behavior has been identified by multiple sites, then counting the unique sites that reaffirm that behavior would generate higher confidence.

3.2. Where Does the Information Come From?

Sites containing information about endpoints could be categorized

- **Web logs:** Many Web servers run Web log analyzer programs such as AWStats, Webalizer, and SurfStats. Such programs collect information about client IP addresses, statistics about access dates, host operating systems and host browsers. They parse the Web server log file and generate a report or a statistics Web page.
- **Proxy logs:** Popular proxy services also generate logs of IP addresses that have accessed them. For instance, the Squid proxy server logs the requests' IP addresses, and then is plays them on a Web page.
- **Forums:** As explained above, Internet forums provide wealth of information about endpoints. Some forums list the user IP addresses along with the user names and the posting dates in order to protect against forum spam. Likewise, very frequently clients use Internet forums to post links containing (often illegal) CDs or DVDs with popular movies as either ftp, http, or streaming shares. We explained above how our methodology captures such cases. Some IP logging forums also provide information about clients participating in chat applications. These forums also ask for (Yahoo, MSN, etc.) messenger ID's upon registration in order to display the online status of the forum user. When searching the IP address on search engine one also finds this related information.

- **Malicious lists:** Denial of service attacks and client misbehavior in general, are a big problem in today's Internet. One of the ways to combat the problem is to track and publicize malicious endpoint behavior. Example lists are: banlists, spamlists, badlists, gaming abuse lists, adserver lists, spyware lists, malware lists, forum spammers lists, etc.
- **Server lists:** For communication to progress in the Internet, information about servers, i.e., which IP address one must contact in order to proceed, must be publicly available. Examples are domain name servers, domain databases, gaming servers, mail servers, IRC servers, router (POP) lists, etc.
- **P2P communication:** In p2p communication, an endpoint can act both as a client and as a server. Consequently, an IP's involvement in p2p applications such as eMule, gnutella, edonkey, kazaa, torrents, p2p streaming software, etc., becomes visible when contacting the first point of entry into the system. For this, this first point of entry is known and typically available on the Web. The number of such endpoints in a p2p network is relatively small; however, whenever a client wants to retrieve a file he typically goes through such an access point. Example Web sites are emule-project.net, edonkey2000.cn, or cache.vagaa.com, that lists torrent nodes.

3.3 Endpoint profiling

Next, we apply this methodology to answer the following questions: i) how can we cluster endpoints that show alike access patterns and how similar or different are these classes for different world regions, and ii) where do clients fetch content from, i.e., how local or international are clients' access patterns for these regions? In all scenarios, we utilize the maximum possible information that we have, and apply our approach accordingly.

A. *Endpoint Clustering*

Algorithm: First, we introduce an algorithm we selected to perform endpoint clustering. The key objective of such clustering is to better understand endpoints' behavior at a large scale in different world regions. Employing clustering in networking has been done before. We select the autoclass algorithm, mainly because it provides unsupervised clustering. This means that, in a Bayesian manner, it can actually infer the different classes from the input data and classify the given inputs with a certain probability into one of these classes. The autoclass algorithm selects the optimal number of classes and also the definition of these classes using a Bayesian maximum posterior probability criterion. In addition to accurate clustering, the algorithm also provides a ranking of the variables according to their significance in generating the classification.

For each of the regions we explore, input to the endpoint clustering algorithm is a set of *tagged* IP addresses from the region's network. Since in this case we are interested in the access behavior of users in the network, we determine the tags via an extension of the mapping in Table IV. For regions with traces, if an *in-network* IP address sends/receives traffic to/from an *out-network* IP address that is tagged by a server tag,

e.g., as , web server then the in-network address is tagged appropriately (using the mapping from column 2 to 3 in the table) as browsing. For regions with no trace (Europe), if an in-network IP address has a client tag found via the endpoint method, then it is tagged via the mapping from column 1 to 3 in the table and we also note the URL of the site where the tag was obtained from. Thus, the in-network IP addresses are tagged as browsing, chat, mail, p2p, ftp, streaming, gaming, malware or combination thereof. The sample set for the explored networks is around 4000 in-network IP addresses for all regions except N. American, where we gather about 21 000 addresses.

A. *Traffic Locality*

Next, we explore where do clients fetch the content from, i.e., how local or global are clients' access patterns? Such patterns might not necessarily reflect clients' interests at the social or cultural levels. For example, a client might access highly 'global' content, generated at another continent, by fetching it from a nearby Content Distribution Network's replica. Likewise, clients can get engaged in a strictly 'local' debate at a forum hosted at the other part of the world. Still, we argue that the results we present below are necessarily affected by clients' interests at social and cultural planes as well.

We proceed as follows. First, from the mechanism mentioned in Section IV-A we obtain a pair of *in-, out-network* IP addresses for each flow. Note that for the case where we only have the URL, we obtain its corresponding IP address via DNS lookup. Next, we obtain the AS-level distance between the two IP addresses by analyzing the BGP Routing Tables as obtained from Routeviews . Finally, we resolve the country code for a given destination AS by using the relevant Internet Routing Registries database (ARIN, RIPE, APNIC, and LACNIC).

4. Internet Profiler Software

Internet Profiler software is my implementation of search engine-based approach for creating profiles of Internet Endpoints. It is a web application and is completely written by Microsoft tools and technologies. I runs on Windows Azure(Microsoft Cloud Server). I use Google search engine and Microsoft Bing to collect information about endpoints. For improving the result, I use the following extra resources: WHOIS servers, Reverse DNS Lookup, P2P systems, and network traces.

Summary of Internet Profiler's specs:

- Operating System : Windows Azure
- Development Environment : Visual Studio 2010
- Programming Language : C#
- Database Management System : Microsoft SQL Azure
- Used Technologies and Application Programming Interfaces(API) : Silverlight, .NET 4, WCF , EF, Prism, LINQ, Parallel LINQ, Google Custom Search API, Bing API, PCAP API
- Client Side Architecture : MVVM design pattern
- Server Side Architecture : SOA design pattern, N-Tire design pattern , DDD pattern , ORM

First, I will explain different APIs used in the software.

4.1. Google Custom Search API

Google Search or **Google Web Search** is a web search engine owned by Google Inc. and is the most-used search engine on the Web Google receives several hundred million queries each day through its various services. The main purpose of Google Search is to hunt for text in webpages, as opposed to other data, such as with Google Image Search. Google search was originally developed by Larry Page and Sergey Brin in 1997.

Google Custom Search enables a developer to create a search engine for a website, a blog, or a software. He can fine-tune the ranking, customize the look and feel of the search results, and invite his friends or trusted users to help him build your custom search engine. Google Custom Search lets you create a customized search experience for your own website. You can use the JSON/Atom Custom Search API to retrieve Google Custom Search results from your custom search engine programmatically.

I use the Google Custom Search API to collect data about an IP address. I created a library based on the API to run query in Google search engine. I retrieve the result in JSON format. The Google search engine has a 100 search quota limit for a day. I you want to do search more than 100, you need to pay 5$ for every 1000 search to Google.

You can retrieve results for a particular search by sending an HTTP GET request to its URI.
The URI for a search has the following format:

https://www.googleapis.com/customsearch/v1?parameters

Three query parameters are required with each search request:

- **API key.** Use the key query parameter to identify your application.
- **Custom search engine identifier.** Use either cx or cref to specify the custom search engine you want to perform this search.
    - Use cx for a search engine created with the Google Custom Search page.
    - Use cref for a linked custom search engine
    - If both are specified, cx is used.
- **Search query.** Use the q query parameter to specify your query.

All other query parameters are optional.

Here is an example search request:

GET https://www.googleapis.com/customsearch/v1?key=INSERT-YOUR-KEY&cx=013036536707430787589:_pqjad5hr1a&q=flowers&alt=json

If the request succeeds, the server responds with a 200 OK HTTP status code and the response data:

200 OK


```
{
 "kind": "customsearch#search",
 "url": {
  "type": "application/json",
  "template":
"https://www.googleapis.com/customsearch/v1?q\u003d{searchTerms}&num\u003d{count?}
&start\u003d{startIndex?}&hr\u003d{language?}&safe\u003d{safe?}&cx\u003d{cx?}&cref\u00
3d{cref?}&sort\u003d{sort?}&alt\u003djson"
 },
 "queries": {
  "nextPage": [
   {
```

```
   "title": "Google Custom Search - flowers",
   "totalResults": 10300000,
   "searchTerms": "flowers",
   "count": 10,
   "startIndex": 11,
   "inputEncoding": "utf8",
   "outputEncoding": "utf8",
   "cx": "013036536707430787589:_pqjad5hr1a"
  }
 ],
 "request": [
  {
   "title": "Google Custom Search - flowers",
   "totalResults": 10300000,
   "searchTerms": "flowers",
   "count": 10,
   "startIndex": 1,
   "inputEncoding": "utf8",
   "outputEncoding": "utf8",
   "cx": "013036536707430787589:_pqjad5hr1a"
  }
 ]
},
 "context": {
  "title": "Custom Search"
 },
 "items": [
 {
   "kind": "customsearch#result",
   "title": "Flower - Wikipedia, the free encyclopedia",
   "htmlTitle": "\u003cb\u003eFlower\u003c/b\u003e - Wikipedia, the free encyclopedia",
   "link": "http://en.wikipedia.org/wiki/Flower",
   "displayLink": "en.wikipedia.org",
   "snippet": "A flower, sometimes known as a bloom or blossom, is the reproductive structure
found in flowering plants (plants of the division Magnoliophyta, ...",
   "htmlSnippet": "A \u003cb\u003eflower\u003c/b\u003e, sometimes known as a bloom or
blossom, is the reproductive structure \u003cbr\u003e  found in flowering plants (plants of the
division Magnoliophyta, \u003cb\u003e...\u003c/b\u003e",
   "pagemap": {
    "RTO": [
     {
      "format": "image",
      "group_impression_tag": "prbx_kr_rto_term_enc",
      "Opt::max_rank_top": "0",
```

```
    "Opt::threshold_override": "3",
    "Opt::disallow_same_domain": "1",
    "Output::title": "\u003cb\u003eFlower\u003c/b\u003e",
    "Output::want_title_on_right": "true",
    "Output::num_lines1": "3",
    "Output::text1": "꽃은 식물 에서 씨 를 만들어 번식 기능을 수행하는 생식 기관 을 말한다. 꽃을 형태학적으로 관찰하여 최초로 총괄한 사람은 식물계를 24강으로 분류한 린네 였다. 그 후 꽃은 식물분류학상 중요한 기준이 되었다.",
    "Output::gray1b": "- 위키백과",
    "Output::no_clip1b": "true",
    "UrlOutput::url2": "http://en.wikipedia.org/wiki/Flower",
    "Output::link2": "위키백과 (영문)",
    "Output::text2b": "  ",
    "UrlOutput::url2c": "http://ko.wikipedia.org/wiki/꽃",
    "Output::link2c": "위키백과",
    "result_group_header": "백과사전",
    "Output::image_url": "http://www.gstatic.com/richsnippets/b/fcb6ee50e488743f.jpg",
    "image_size": "80x80",
    "Output::inline_image_width": "80",
    "Output::inline_image_height": "80",
    "Output::image_border": "1"
   }
  ]
 }
 },
 ...
]
}
```


Response data

As shown in the sample output above, the response data includes three main classes of properties:

- Search metadata
- Custom search engine metadata
- Search results

Search metadata

The search metadata includes the url property, which has information about the OpenSearch template used for the results returned in this request.

It also includes a queries property, which is an array of objects describing the characteristics of possible searches. The name of each object in the array is either the name of an OpenSearch query role or one of the two custom roles defined by this API: previousPage and nextPage.

Each query role object is itself an array, though usually the array contains a single element. Possible query role objects include:

- request: Metadata describing the query for the current set of results.
  - This role is always present in the response.
  - It is always an array with just one element.
- nextPage: Metadata describing the query to use for the next page of results.
  - This role is not present if the current results are the last page. **Note:** This API returns up to the first 100 results only.
  - When present, it is alwas a array with just one element.
- previousPage: Metadata describing the query to use for the previous page of results.
  - Not present if the current results are the first page.
  - When present, it is always a array with just one element.

Custom search engine metadata

The context property has metadata describing the particular search engine that performed the search query. It includes the name of the search engine, and any facet objects it provides for refining a search.

Search results

The items array contains the actual search results. The search results include the URL, title and text snippets that describe the result. In addition, they can contain pagemap information, if applicable.

If the search results include a promotions property, it contains a set of subscribed links results.

Optional query parameters

Data format

You can select the data format of the returned results with the alt query parameter.

Pagination

This API can return the first 100 search results, in chunks (pages) up to 10 search results long. You can specify the page of search results to return using the start and num query parameters.

Result restrictions

You can restrict the search results returned in the following ways:

- **Document language -** use the lr query parameter.
- **Safe search -** use the safe query parameter.

4.2. Microsoft Bing API

Bing (formerly **Live Search**, **Windows Live Search**, and **MSN Search**) is a web search engine (advertised as a "decision engine" from Microsoft. Bing was unveiled by Microsoft CEO Steve Ballmer on May 28, 2009 at the *All Things Digital* conference in San Diego. It went fully online on June 3, 2009, with a preview version released on June 1, 2009.

Notable changes include the listing of search suggestions as queries are entered and a list of related searches (called "Explorer pane") based on semantic technology from Powerset that Microsoft purchased in 2008.

Before the launch of Bing, the marketshare of Microsoft web search pages (MSN and Live search) had been stagnant. By January 2011, Experian Hitwise show that Bing's market share had increased to 12.8% at the expense of Yahoo and Google. Bing powered searches also continued to have a higher "success rate" compared to Google, with more users clicking on the resulting links.[ ] In the same period, comScore's "2010 U.S. Digital Year in Review" report showed that "Bing was the big gainer in year-over-year search activity, picking up 29% more searches in 2010 than it did in 2009." The Wall Street Journal notes the 1% jump in share "appeared to come at the expense of rival Google Inc". In February 2011 Bing beat out Yahoo! for the first time ever in terms of search marketshare. Bing received 4.37% search share while Yahoo! received 3.93% according to StatCounter.

I use the Bing API to collect data about an IP address. Bing returns data in JSON, XML, and SOAP formats. I created a library based on the API to run query in Google search engine. I retrieve the result in JSON format. The Bing does not have quota limit. The only limitation is from an IP address only 7 queries can be run in a second.

Using JSON (Bing, Version 2)

Bing Services

The Bing JSON interface is an HTTP GET interface which accepts search requests in URL format and returns search results in JSON format.

Sending a Request in URL Format

In order to use the JSON interface, you need to know is how to submit a search request in URL format.When a search request is submitted in URL format, each URL parameter presents a field

in the search request. A simple search request that returns a Web result and a Spell result could look like this:

http://api.bing.net/json.aspx?AppId=*YOUR_APPID*&Version=2.2&Market=en-US&Query=testign&Sources=web+spell&Web.Count=1

AppId, Version, Market, Query, and Sources are the first-level fields of SearchRequest object, which can be directly specified as URL parameters. Simple types like AppId can be specified by the syntax *Field*=*Value*. However, array types such as Sources, the list of SourceType enumerations, should be separated by +. Thus, the syntax will look like *Field*=*Value1*+*Value2*+…+*ValueN*. In addition, to specify the value of fields in a structure type, like Count in WebRequest, the . operator is used to access sub-fields with the syntax *Field*.*SubField*=*Value*.

Note: Arrays of arrays or structures are not supported in the request interface of Bing API 2.0. For arrays of strings, space in each string must be encoded. For example, use %20 as opposed to +.

For details of search request and response objects, refer to the following topics:

- SearchRequest Class (Bing, Version 2.0)
- SearchResponse Class (Bing, Version 2.0)

Using JSONType

For the JSON interface, there is an interface-specific parameter JsonType, which can control the format of the response. If raw enumeration is specified, search results are returned in pure JSON format. If callback enumeration is specified, a JavaScript statement will be returned to call the callback function specified in the JsonCallback parameter and search results will be passed in as arguments. If the function enumeration is specified, a JavaScript function will be returned and the search results will be returned when the function is invoked. The following sections give an example request and response for each of these options.

Raw Enumeration Example

Request

http://api.bing.net/json.aspx?AppId=***YOUR_APPID***&Version=2.2&Market=en-US&Query=testign&Sources=web+spell&Web.Count=1&JsonType=raw

Note: For information about obtaining an AppId, see Bing Developer Center.

Response


```
{
  "SearchResponse":{
    "Version":"2.2",
    "Query":{
      "SearchTerms":"testign"
    },
    "Spell":{
      "Total":1,
      "Results":[
        {
          "Value":"testing"
        }
      ]
    },
    "Web":{
      "Total":5100,
      "Offset":0,
      "Results":[
        {
          "Title":"Testign part 2 - Tiernan OTooles Programming Blog",
          "Description":"If this works, it means nothing really, but i have man
ged to build a .TEXT blog posting app. could be handy if i move my main blog to .TEXT, which i a
m thinking about..",
          "Url":"http:\/\/weblogs.asp.net\/tiernanotoole\/archive\/2004\/09\/24\/233830.aspx"
,
          "DisplayUrl":"http:\/\/weblogs.asp.net\/tiernanotoole\/archive\/2004\/09\/24\/23383
0.aspx",
          "DateTime":"2008-10-21T05:08:05Z"
        }
      ]
    }
  }
}
```

Function Enumeration Example

Request

http://api.bing.net/json.aspx?AppId=***YOUR_APPID***&Version=2.2&Market=en-US&Query=testign&Sources=web+spell&Web.Count=1&JsonType=function

Note: For information about obtaining an AppId, see Bing Developer Center.

Response


```
function LiveSearchGetResponse(){
  return{
    "SearchResponse":{
      "Version":"2.2",
      "Query":{
        "SearchTerms":"testign"
      },
      "Spell":{
        "Total":1,
        "Results":[
          {
            "Value":"testing"
          }
        ]
      },
      "Web":{
        "Total":5100,
        "Offset":0,
        "Results":[
          {
            "Title":"Testign part 2 - Tiernan OTooles Programming Blog",
            "Description":"If this works, it means nothing really, but i have managed to build a .TEXT blog posting app. could be handy if i move my main blog to .TEXT, which i am thinking about..",
            "Url":"http:\/\/weblogs.asp.net\/tiernanotoole\/archive\/2004\/09\/24\/233830.aspx",
            "DisplayUrl":"http:\/\/weblogs.asp.net\/tiernanotoole\/archive\/2004\/09\/24\/233830.aspx",
```

```
            "DateTime":"2008-10-21T05:08:05Z"
          }
        ]
      }
    }/*pageview_candidate*/
  };
}
```


Callback Enumeration Example

Request

http://api.bing.net/json.aspx?AppId=***YOUR_APPID***&Version=2.2&Market=en-US&Query=testign&Sources=web+spell&Web.Count=1&JsonType=callback&JsonCallback=UserCallback

Note: For information about obtaining an AppId, see Bing Developer Center.

Response


```
if(typeofUserCallback=='function')UserCallback({
  "SearchResponse":{
    "Version":"2.2",
    "Query":{
      "SearchTerms":"testign"
    },
    "Spell":{
      "Total":1,
      "Results":[
        {
          "Value":"testing"
        }
      ]
    },
    "Web":{
      "Total":5100,
      "Offset":0,
      "Results":[
        {
          "Title":"Testign part 2 - Tiernan OTooles Programming Blog",
```

"Description":"If this works, it means nothing really, but i have managed to build a .TEXT blog posting app. could be handy if i move my main blog to .TEXT, which i am thinking about..",

"Url":"http:\/\/weblogs.asp.net\/tiernanotoole\/archive\/2004\/09\/24\/233830.aspx",

"DisplayUrl":"http:\/\/weblogs.asp.net\/tiernanotoole\/archive\/2004\/09\/24\/233830.aspx",

"DateTime":"2008-10-21T05:08:05Z"

}

]

}

}/*pageview_candidate*/

});

3.3. WHOIS server

NICNAME/WHOIS server is a TCP transaction based query/response server, running on a few specific central machines, that provides net wide directory service to Internet users. The Network Information Center (NIC) maintains the central NICNAME database and server, defined in RFC 954, providing online look-up of individuals, network organizations, key host machines, and other information of interest to users of the Internet. Many sites now maintain local directory servers with information about individuals, departments and services at that specific site.

4.3. WHOIS SERVER FOR IP ADDRESS

A regional Internet registry (RIR) is an organization that manages the allocation and registration of Internet number resources within a particular region of the world. Internet number resources include IP addresses and autonomous system (AS) numbers.
There are five RIRs:

- African Network Information Centre (AfriNIC) for Africa
- American Registry for Internet Numbers (ARIN) for the United States, Canada, and several parts of the Caribbean region.
- Asia-Pacific Network Information Centre (APNIC) for Asia, Australia, New Zealand, and neighboring countries
- Latin America and Caribbean Network Information Centre (LACNIC) for Latin America and parts of the Caribbean region
- Réseaux IP Européens Network Coordination Centre (RIPE) for Europe, the Middle East , and Central Asia

I use the WHOIS servers as extra resource to gather information about IP addresses. WHOIS service runs on well-known TCP/43 port in a server. I created a library to connect to this port and collect the data such as networks, name servers, and autonomous system numbers and Points of Contact.

4.3. WINPCAP LIB

WinPcap is the industry-standard tool for link-layer network access in Windows environments: it allows applications to capture and transmit network packets bypassing the protocol stack, and has additional useful features, including kernel-level packet filtering, a network statistics engine and support for remote packet capture. WinPcap consists of a driver, that extends the operating system to provide low-level network access, and a library that is used to easily access the low-level network layers. This library also contains the Windows version of the well-known libpcap Unix API.

Thanks to its set of features WinPcap is the packet capture and filtering engine of many open source and commercial network tools, including protocol analyzers, network monitors, network intrusion detection systems, sniffers, traffic generators and network testers. Some of these tools, like Wireshark, Nmap, Snort, ntop are known and used throughout the networking community.

WinPcap is written in c++ . For using WinPcap in .NET platform with c# language, I used Pcap.Net which is a .NET wrapper for WinPcap written in C++/CLI and C#. It Features almost all WinPcap features and includes a packet interpretation framework.

I use Pcap.Net to open trace files to read the packets and trace analysis. This library does not interpret application level protocols such as BGP, so I need add these features to the library based on necessity. I use network traces as extra resources to collect more information about IP addresses.

4.4. Internet Profiler Application

As shown in fig. 6 Internet Profiler running on Windows Azure, somewhere in Internet. It uses different services provided by Windows Azure such as SQL Azure service, .NET service.
I has two main part named as **Internet Provider Service** and **Internet Provider App**.

Internet Provider Service does the following tasks :

- Executes network profiling processes such as query search engines, query WHOIS servers, crawling into P2P networks, collect data from network traces, IP tagging
- Statistical and analytical analysis
- etc.

Internet Provider App provides the user interface to work with **Internet Provider Service.** It will be downloaded into browser on user PC to run. It does the following Tasks :

- Create network profiling
- View result from a network profiling
- Create IP tagging tables
- View statistical and analytical analysis results in graphs and charts
- View traces
- etc.

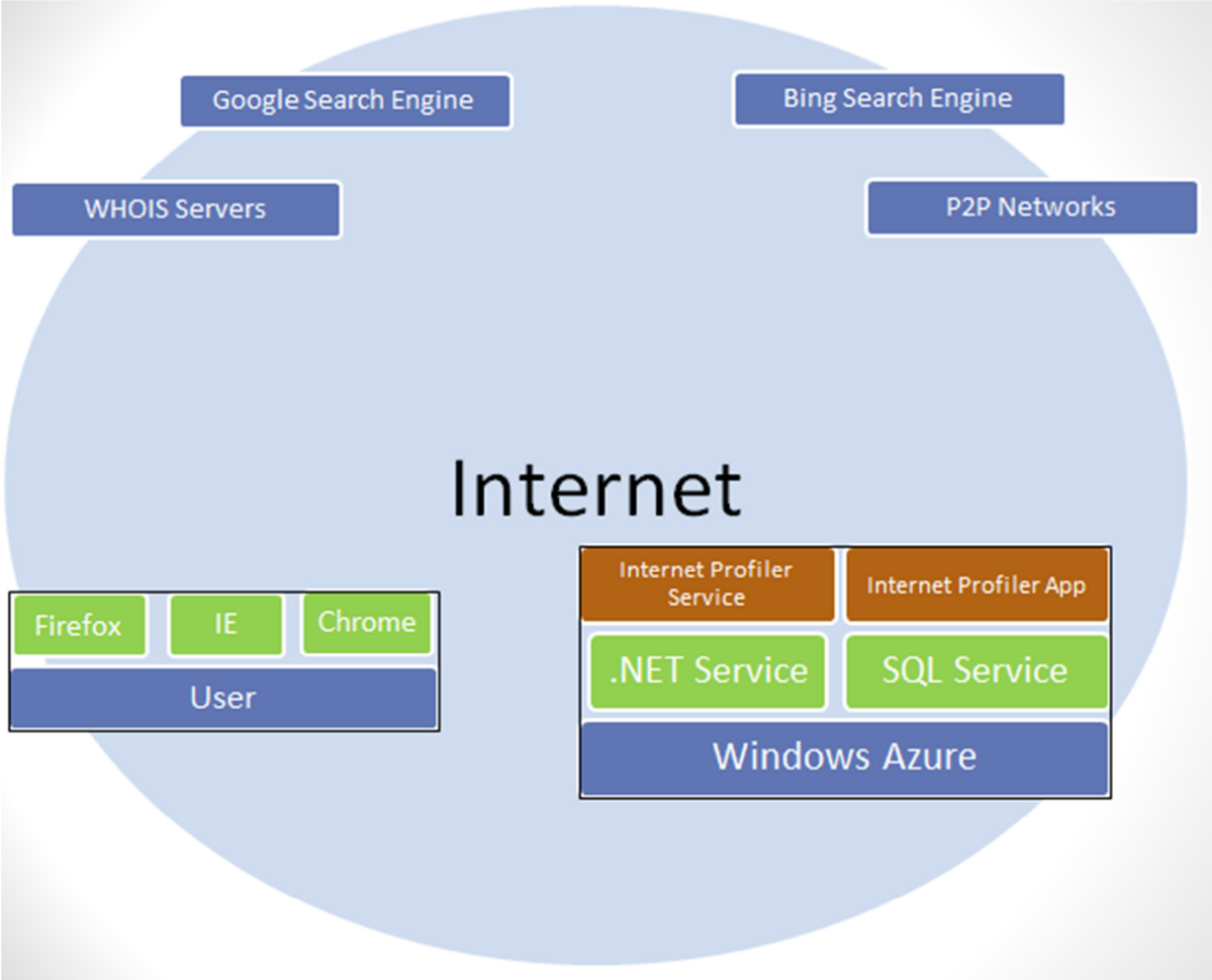

Fig 6: Visual representation of Internet Profiler software and related components

5. References:

[1] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, “BLINC: Multilevel traffic classification in the dark,” in *Proc. ACM SIGCOMM*, Philadelphia, PA, Aug. 2005, pp. 229–240.

[2] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, “Internet traffic classification demystified: Myths, caveats, and the best practices,” in *Proc. ACM CONEXT*, Madrid, Spain, Dec. 2008, Article no. 11.

[3] T. Karagiannis, K. Papagiannaki, N. Taft, and M. Faloutsos, “Profiling the end host,” in *Proc. PAM*, Louvain-la-neuve, Belgium, Apr. 2007, pp. 186–196.

[4] I. Trestian, S. Ranjan, A. Kuzmanovi, and A. Nucci, “Googling the internet: profiling internet endpoints via the world wide web” in *Proc. ACM SIGCOMM*, New York, NY, USA 2008, pp. 666-679.

[5] Google Custom Search, [Online]. Available: http://code.google.com/apis/customsearch/

[6] Bing API, [Online]. Available: http://www.bing.com/developers/

[7] PCAP.NET, [Online], Available: http://pcapdotnet.codeplex.com/

[8] WinPCAP, [Online], Available: http://www.winpcap.org/

[9] Wikipedia, [Online], Available: http://www.wikipedia.org/