DNS to the Rescue: Discerning Content and Services in a Tangled Web

Ignacio N. Bermudez
Politecnico di Torino,
Torino, Italy
ignacio.bermudez@polito.it

Marco Mellia
Politecnico di Torino,
Torino, Italy
marco.mellia@polito.it

Maurizio M. Munafò
Politecnico di Torino,
Torino, Italy
maurizio.munafo@polito.it

Ram Keralapura
Narus Inc.,
Sunnyvale, CA
rkeralapura@narus.com

Antonio Nucci
Narus Inc.,
Sunnyvale, CA
anucci@narus.com

ABSTRACT

A careful perusal of the Internet evolution reveals two major trends - explosion of cloud-based services and video streaming applications. In both of the above cases, the owner (e.g., CNN, YouTube, or Zynga) of the content and the organization serving it (e.g., Akamai, Limelight, or Amazon EC2) are decoupled, thus making it harder to understand the association between the content, owner, and the host where the content resides. This has created a tangled world wide web that is very hard to unwind, impairing ISPs' and network administrators' capabilities to control the traffic flowing in their networks.

In this paper, we present DN-Hunter, a system that leverages the information provided by DNS traffic to discern the tangle. Parsing through DNS queries, DN-Hunter tags traffic flows with the associated domain name. This association has several applications and reveals a large amount of useful information: (*i*) Provides a fine-grained traffic visibility even when the traffic is encrypted (i.e., TLS/SSL flows), thus enabling more effective policy controls, (*ii*) Identifies flows even before the flows begin, thus providing superior network management capabilities to administrators, (*iii*) Understand and track (over time) different CDNs and cloud providers that host content for a particular resource, (*iv*) Discern all the services/content hosted by a given CDN or cloud provider in a particular geography and time interval, and (*v*) Provides insights into all applications/services running on any given layer-4 port number.

We conduct extensive experimental analysis and show results from real traffic traces (including FTTH and 4G ISPs) that support our hypothesis. Simply put, the information provided by DNS traffic is one of the key components required for understanding the tangled web, and bringing the ability to effectively manage network traffic back to the operators.

IMC'12, November 14–16, 2012, Boston, Massachusetts, USA.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Miscellaneous; C.4 [**Performance of Systems**]: Measurement Techniques

General Terms

Measurement, Performance

Keywords

DNS, Service Identification.

1. INTRODUCTION

In the past few years, the Internet has witnessed an explosion of cloud-based services and video streaming applications. In both cases, content delivery networks (CDN) and/or cloud computing services are used to meet both scalability and availability requirements. An undesirable side-effect of this is that it decouples the owner of the content and the organization serving it. For example, CNN or YouTube videos can be served by Akamai or Google CDN, and Farmville game can be accessed from Facebook while running on Amazon EC2 cloud computing platform, with static content being retrieved from a CDN. This may be even more complicated since various CDNs and content owners implement their own optimization mechanisms to ensure "spatial" and "temporal" diversity for load distribution. In addition, several popular sites like Twitter, Facebook, and Google have started adopting encryption (TLS/SSL) to deliver content to their users [1]. This trend is expected to gain more momentum in the next few years. While this helps to protect end-users' privacy, it can be a big impediment for effective security operations since network/security administrators now lack the required traffic visibility. The above factors have resulted in "tangled" world wide web which is hard to understand, discern, and control.

In the face of this tangled web, network/security administrators seek answers for several questions in order to manage their networks: (*i*) What are the various services/applications that contribute to the traffic mix on the network? (*ii*) How to block or provide certain Quality of Service (QoS) guarantees to select services?

While the above questions seem simple, the answers to these questions are non-trivial. There are no existing mechanisms that can provide comprehensive solutions to address the above issues. Consider the first question above. A typical approach currently used by network administrators is to rely on DPI (deep packet inspection) technology to identify traffic based on packet-content signatures. Although this approach is very effective in identifying unencrypted traffic, it severely falls short when the traffic is encrypted. Given the popularity of TLS in major application/content providers, this problem will amplify over time, thus rendering typical DPI technology for traffic visibility ineffective. A simple approach that can augment a DPI device to identify encrypted traffic is to inspect the certificate during the initial handshake[1]. Although this approach gives some visibility into the applications/services, it still cannot help in identifying specific services. For instance, inspecting a certificate from Google will only reveal that it is Google service, but cannot differentiate between Google Mail, Google Docs, Blogger, and Youtube. Thus administrators need a solution that will provide fine-grained traffic visibility even when the traffic is encrypted.

Let us now focus on the second question which is even more complex. Consider the scenario where the network administrator wants to block all traffic to Zynga games, but prioritize traffic for the DropBox service. Notice that both of these services are encrypted, thus severely impairing a DPI-based solution. Furthermore, both of these services use the Amazon EC2 cloud. In other words, the server IP-address for both of these services can be the same. Thus using IP-address filtering does not accomplish the task either. In addition the IP-address can change over time according to CDN optimization policies. Another approach that can be used in this context is to introduce certain policies directly into the local name servers. For example, the name server does not resolve the DNS query for zynga.com in the above example, thus blocking all traffic to Zynga. Although this approach can work effectively for blocking certain services, it does not help when administrators are interested in prioritizing traffic to certain services. Administrators face the same situation when they want to prioritize traffic to *mail.google.com* and *docs.google.com*, while de-prioritizing traffic *blogspot.com* and *youtube.com* since all of these services can run over HTTPS on the same Google platform.

In this work, we propose DN-Hunter, a novel traffic monitoring system that addresses all of the above issues in a completely automated way. The main intuition behind DN-Hunter is to correlate the DNS queries and responses with the actual data flows in order to effectively *identify* and *label* the data flows, thus providing a very fine grained visibility of traffic on a network. It helps network administrators to keep track of the mapping between users, content owners, and the hosts serving the content even when this mapping is changing over time, thus enabling them to enforce policies on the traffic at any time with no manual intervention. In addition, network administrators could use DN-Hunter to dynamically reroute traffic in order to use more cost-effective links (or high bandwidth links as the policies might dictate) even as the content providers change the hosts serving the content over time for load balancing or other economic reasons.

At a high level, the methodology used in DN-Hunter seems to be achievable by performing a simple reverse DNS lookup using the server IP-addresses seen in traffic flows. However, using reverse DNS lookup does not help since it does not return accurate domain (or the sub-domain) names used in traffic flows.

The main contributions of this work are:

- We propose a novel tool, DN-Hunter, that can provide *fine-grained traffic visibility* to network administrators for effective policy controls and network management. Unlike DPI technology, using experiments on real traces, we show that DN-Hunter is very effective even when the traffic is encrypted clearly highlighting its advantages when compared to the current approaches. DN-Hunter can be used either for active or passive monitoring, and can run either as a stand-alone tool or can easily be integrated into existing monitoring systems, depending on the final intent.
- A key property of DN-Hunter is its ability to identify traffic *even before the data flow starts*. In other words, the information extracted from the DNS responses can help a network management tool to *foresee* what kind of flows will traverse the network. This unique ability can empower proactive traffic management policies, e.g., prioritizing all TCP packets in a flow (including the critical three-way-handshake), not just those packets that follow a positive DPI match.
- We use DN-Hunter to not only provide real-time traffic visibility and policy controls, but also to help gain better understanding of how the dynamic web is organized and evolving today. In other words, we show many other applications of DN-Hunter including: (*i*) *Spatial Discovery:* Mapping a particular content to the servers that actually deliver them at any point in time. (*ii*) *Content Discovery:* Mapping all the content delivered by different CDNs and cloud providers by aggregating the information based on server IP-addresses. (*iii*) *Service Tag Extraction:* Associating a layer-4 port number to the most popular service seen on the port with no a-priori information.
- We conduct extensive experiments using five traffic traces collected from large ISPs in Europe and North America. The traces contain full packets including the application payload, and range from 3h to 24h. These ISPs use several different access technologies (ADSL, FTTH, and 3G/4G) to provide service to their customers, thus showing that DN-Hunter is effective in several different contexts. Furthermore, DN-Hunter has been implemented and currently deployed in three operative vantage points since March 2012.

Although DN-Hunter is a very effective tool in any network administrator's arsenal to address issues that do not have a standard solution today, there are some limitations as well. First, the effectiveness of DN-Hunter depends on the visibility into the DNS traffic of the ISP/enterprise. In other words, DN-Hunter will be rendered useless if it does not have visibility into the DNS queries and responses along with the data flows from the end-users. Second, DN-Hunter does not help in providing visibility into applications/services that do not depend on DNS. For instance, some peer-to-peer applications are designed to work with just IP-addresses and DN-Hunter will be unable to label these flows. Third, automatic

[1] During TLS negotiation, the server certificate contains a plain text string with the name being signed.

Table 1: Dataset description.

Trace	Start [GMT]	Duration	Peak DNS Responses Rate	#Flows TCP
US-3G	15:30	3h	7.5k/min	4M
EU2-ADSL	14:50	6h	22k/min	16M
EU1-ADSL1	8:00	24h	35k/min	38M
EU1-ADSL2	8:40	5h	12k/min	5M
EU1-FTTH	17:00	3h	3k/min	1M

and smart algorithms must be devised to dig into the information exposed by DN-Hunter. In this paper, we provide some examples of how the information extracted from the DNS traffic can be used. We believe that the applications of DN-Hunter are not limited to the ones presented in this work, and novel applications can leverage the information exposed by DN-Hunter.

The rest of the paper is organized as follows: Sec. 2 introduces the datasets we use in this paper. In Sec. 3 we describe the architecture and design details of DN-Hunter. Sec. 4 presents some of our advanced analytics modules while Sec. 5 provides extensive experimental results. We discuss correct dimensioning and deployment issues in Sec. 6. We highlight the major differences between DN-Hunter and some existing approaches in Sec. 7 and conclude the paper in Sec. 8.

2. DATASETS AND TERMINOLOGY

In this section, we provide insight into the datasets used for experimental evaluation along with some basic DNS terminology used henceforth in this paper.

2.1 Experimental datasets

All our datasets are collected at the Points-of-Presence (PoP) of large ISPs where the end customers are connected to the Internet. The five datasets we use in this paper are reported in Tab. 1. In all of these traces activities from several thousands of customers are monitored. In all the 5 datasets, we capture full packets including the application payload without any packet losses. For the sake of brevity, Tab. 1 only reports the start time and trace duration, the peak time DNS response rate, and the number of TCP flows that were tracked. Each trace corresponds to a different period in 2011. The first dataset is a trace collected from a large North American 3G/4G mobile operator GGSN aggregating traffic from a citywide area. The second dataset originates from a European ISP (EU2) which has about 10K customers connected via ADSL technology. The last three datasets correspond to traffic collected from different vantage points in the same European ISP (EU1). The vantage points are located in three different cities - two ADSL PoPs and one Fiber-To-The-Home (FTTH) access PoP.

Currently, DN-Hunter has been implemented in a commercial tool as well as *Tstat* [2]. The latter has been deployed in all the three vantage points in EU1 and has been successfully labeling flows since March 2012. Some of the results in this paper are derived from this deployment.

2.2 DNS Terminology

DNS is a hierarchical distributed naming system for computers connected to the Internet. It translates "domain names" that are meaningful to humans into IP-addresses

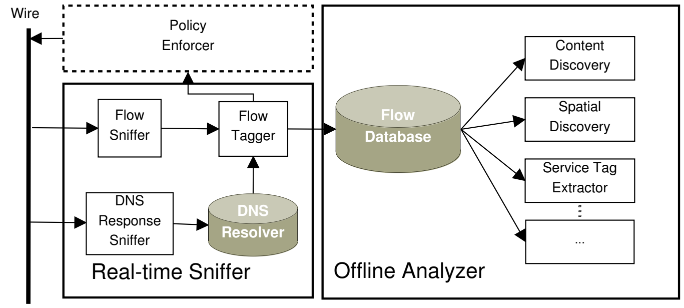

Figure 1: DN-Hunter architecture overview

required for routing. A DNS name server stores the DNS records for different domain names.

A domain name consists of two or more "labels" that are conventionally concatenated, and delimited by dots, e.g., *www.example.com*. These names provide meaningful information to the end user. Therefore labels naturally convey information about the service, content, and information offered by a given domain name. The labels in the domain name are organized in a hierarchical fashion. The Top-Level Domain (TLD) is the last part of the domain name - *.com* in the above example; and sub-domains are then pre-pended to the TLD. Thus, *example.com* is a subdomain of *.com*, and *www.example.com* is a subdomain of *example.com*. In this paper, we refer to the first sub-domain after the TLD as "second level domain"; it generally refers to the organization that owns the domain name (e.g., *example.com*). Finally *Fully Qualified Domain Name* (FQDN) is the domain name complete with all the labels that unambiguously identifies a resource, e.g., *www.example.com*.

When an application needs to access a resource, a query is sent to the local DNS server. This server responds back with the resolution if it already has one, else it invokes an iterative address resolution mechanism until it can resolve the domain name (or determine that it cannot be resolved). The responses from the DNS server carry a *list of answers*, i.e., a list of $serverIP$ addresses that can serve the content for the requested resource.

Local caching of DNS responses at the end-hosts is commonly used to avoid initiating new requests to the DNS server for every resolution. The time for which a local cache stores a DNS record is determined by the Time-To-Live (TTL) value associated with every record. It is set by the authoritative DNS name server, and varies from few seconds (e.g., for CDNs and highly dynamic services) to days. Also, memory limits and timeout deletion policies can affect local caching at the client OS. However, as we will see later, in practice, clients cache DNS responses for typically less than 1 hour.

3. DN-Hunter ARCHITECTURE

A high level overview of DN-Hunter architecture is shown in Fig. 1. It consists of two main components: *real-time sniffer* and *off-line analyzer*. As the name indicates, the sniffer labels/tags all the incoming data flows in *real time*. The output from the sniffer can be used for online policy enforcement (using any available policy enforcing tool) and/or can be stored in a database for off-line analysis by the analyzer component. Note that the sniffer can be a passive

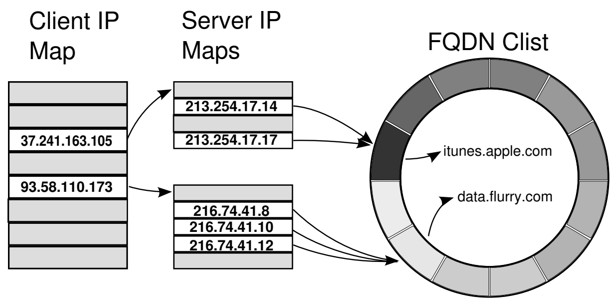

Figure 2: DNS Resolver data structures

```
1: INSERT(DNSresponse)
2: Input: DNSresponse
3: (FQDN, ClientIP, answerList) ← decode(DNSresponse)
4: DNEntry ← newDNEntry(FQDN)
5: mapServer ← mapClient.get(clientIP)
6: if mapServer = null then
7:    mapServer ← new MapServer()
8:    mapClient.put(clientIP, mapServer)
9: end if
10: for all serverIP in answerList do
11:    /* replace old references */
12:    if exists mapSever.get(serverIP) then
13:       OLDEntry ← mapSever.get(serverIP)
14:       OLDEntry.removeOldReferences()
15:    end if
16:    /* Link back and forth
17:    references to the new DNSEntry */
18:    mapServer.put(serverIP, DNEntry)
19:    MSEntry ← mapServer.get(serverIP)
20:    DNEntry.insert(MSEntry)
21: end for
22: /* insert next entry in circular array */
23: OldDNEntry ← Clist.nextEntry()
24: OldDNEntry.deleteBackreferences()
25: Clist.nextEntry ← DNEntry
26:
27: LOOKUP(ClientIP, ServerIP)
28: Input: ClientIP and ServerIP of a flow
29: Output: FQDN of ServerIP as requested by ClientIP
30: mapServer ← mapClient.get(clientIP)
31: if mapServer contains serverIP then
32:    DNEntry ← mapServer.get(serverIP)
33:    return DNEntry.FQDN
34: end if
```

Algorithm 1: DNS Resolver pseudo-code

component instead of being active if the policy enforcer is not implemented. For the ease of exposition, in this work, we assume that the real-time sniffer component is a passive monitoring component.

3.1 Real-Time Sniffer Component

The sniffer has two low-level sniffing blocks: (i) *Flow sniffer* which reconstructs layer-4 flows by aggregating packets based on the 5-tuple $Fid = (clientIP, serverIP, sPort, dPort, protocol)$, and ($ii$) *DNS response sniffer* which decodes the DNS responses, and maintains a local data structure called the *DNS Resolver*. The DNS resolver maintains a mapping between client IP, domain names queried, and the server IP(s) included in the DNS response. In particular, for each response, it stores the set of $serverIP$ addresses returned for the fully qualified domain name (FQDN) queried, associating them to the $clientIP$ that generated the query.

All data flows reconstructed by the flow sniffer is passed on to the *Flow Tagger* module. The flow tagger module queries the DNS resolver to tag the incoming $clientIP, serverIP$ pair. The flow tagger will tag the incoming flow with the "label" (i.e., the FQDN) and sends the flow to the policy enforcer (to enforce any policy on the flow including blocking, redirection, rate limiting, etc.) and/or the database for off-line analysis.

3.1.1 DNS Resolver Design

The key block in the real-time sniffer component is the DNS Resolver. Its engineering is not trivial since it has to meet real-time constraints. The goal of the DNS Resolver is to build a replica of the client DNS cache by sniffing DNS responses from the DNS server. Each entry in the cache stores the $FQDN$ and uses the $serverIP$ and $clientIP$ as look-up keys. To avoid garbage collection, $FQDN$s are stored in a first-in-first-out FIFO circular list, $Clist$, of size L; a pointer identifies the next available location where an entry can be inserted. L limits the cache entry lifetime and has to properly match the local resolver cache in the monitored hosts.

Lookup is performed using two sets of tables. The first table uses the $clientIP$ as key to find a second table, from where the $serverIP$ key points to the most recent $FQDN$ entries in the $Clist$ that was queried by $clientIP$. Tables are implemented using C++ maps[2], in which the elements are sorted from lower to higher key value following a specific strict weak ordering criterion based on IP addresses. Let N_C and $N_S(c)$ represent the number of monitored clients and the number of servers that client c contacts, respectively. Assuming L is well-dimensioned, the look-up complexity is $O(\log(N_C) + \log(N_S(c)))$. N_C depends on the number of hosts in the monitored network. $N_S(c)$ depends on the traffic generated by clients. In general, $N_S(c)$ is in the order of a few hundreds. Note that when the number of monitored clients increase, several load balancing strategies can be used. For example, two resolvers can be maintained for odd and even fourth octet value in the client IP-address.

Fig. 2 depicts the internal data structures in the DNS resolver. Algorithm 1 provides the pseudo code of the "insert()" and "lookup()" functions that access the data structures in Fig. 2. Since DNS responses carry a list of possible $serverIP$ addresses, more than one $serverIP$ can point to the same $FQDN$ entry (line 11-22). When a new DNS response is observed, the information is inserted in the $Clist$, eventually removing old entries (line 12-15)[3]. When an entry in the DNS circular array is overwritten, the old $clientIP$ and $serverIP$ keys are removed from the maps before inserting the new one (line 25).

3.1.2 DNS traffic characteristics

Using the above algorithm for tagging (or labeling) incoming data flows, we conducted several experiments to accomplish the following goals: (i) Understand how much information DNS traffic can expose in enabling traffic visibility, and (ii) Understand how to correctly dimension the DNS resolver data structures.

[2]Unordered maps, i.e., hash tables, can be used as well to further reduce the computational costs

[3]In this case the information about the old FQDN is lost and may create some ambiguity. See Sec. 6 for more details.

Protocol	EU1-ADSL1	EU1-ADSL2	EU1-FTTH
HTTP	92% (4.4M)	90% (2.7M)	91% (683k)
TLS	92% (0.4M)	86% (196k)	84% (50k)
P2P	1% (6k)	1% (1.3k)	0% (48)
	EU2-ADSL	**US-3G**	
HTTP	97% (5.8M)	75% (445k)	
TLS	96% (279k)	74% (83k)	
P2P	1% (4.2k)	8% (8k)	

Table 2: DNS Resolver hit ratio

To address the first goal, we compute the DNS hit ratio. In other words, DNS hit ratio represents the fraction of data flows that can be successfully associated with a FQDN. The higher is the hit ratio, the more successful is DN-Hunter in enabling traffic visibility. Intuition suggests that all client-server services/applications rely on the DNS infrastructure and hence DN-Hunter will be able to accurately identify them. However, certain peer-to-peer services/applications do not use the DNS infrastructure and thus evade detection in DN-Hunter. Tab. 2 confirms this intuition. It details, for each trace, the number of DNS hits and the corresponding percentage of flows that were resolved, considering the subset of HTTP, TLS, and P2P flows. In this experiment, we consider a warm-up time of 5 minutes (i.e., we track all flows, but ignore the statistics contributed by the flows in the first 5 mins of the trace).

As expected, HTTP and TLS flows show a very high hit ratio, with the majority of cache-miss occurring in the initial part of the trace when the end host operating system local resolver cache resolves the query locally and limits the queries to the DNS server. P2P data flows are hardly preceded by DNS resolutions, and hence it results in a very low hit ratio[4].

When considering only HTTP and TLS data flows, we see that the hit ratio mostly exceeds 90% for all traces except US-3G. When considering only the last hour of each trace, the DNS hit ratio increases further close to 100% in all traces but US-3G. In the case of US-3G, we hypothesize that the adoption of tunneling mechanisms over HTTP/HTTPS for which no DNS information is exposed may be the cause of lower DNS Resolver efficiency. Furthermore, device mobility may also affect our results: our tool may observe flows from devices entering the coverage area after performing a DNS resolution outside the visibility of our monitoring point. Thus our tool might miss the DNS response resulting in a cache-miss. More details about the DNS traffic characteristics that affects DN-Hunter dimensioning is provided in Sec. 6.

3.1.3 *DN-Hunter vs. DNS reverse lookup*

The information that the sniffer component extracts is much more valuable than the one that can be obtained by performing active DNS reverse lookup of $serverIP$ addresses. Recall that the reverse lookup returns only the designated domain name record. Consider Tab. 3 where we randomly selected 1,000 $serverIP$ for which the Sniffer was able to associate a FQDN. We have considered the EU1-ADSL2 dataset for this experiment. We then performed active DNS reverse lookup queries of the $serverIP$ addresses and compared the returned FQDN with the one recovered by the sniffer. In 29% of cases, no answer was returned by the reverse lookup while in 26% of the lookups the two answers were totally different from each other. All the other queries had at least had a partial match. In fact, only 9% of the reverse lookups completely matched the results from the sniffer while the rest of the 36% only matched the second-level domain name. These results are not surprising since single servers are typically serving several FQDNs (see Sec. 5). In addition to this, reverse lookup poses scalability issues as well.

Same FQDN	9%
Same 2nd-level domain	36%
Totally different	26%
No-answer	29%

Table 3: DN-Hunter vs. reverse lookup

3.2 Off-Line Analyzer Component

Although the sniffer module provides deep visibility into the services/applications on the wire in real-time, some analytics cannot be performed in real-time. In other words, dissecting and analyzing the data in different ways can expose very interesting insights about the traffic. The off-line analyzer component does exactly this. It contains several intelligent analytics that can extract information from the flows database by mining its content. In the next section, we will present a few sample analytics. However, several other analytics can be added into the system easily.

4. ADVANCED ANALYTICS

In this section we describe some advanced analytics using the data stored in the labeled flows database to automatically discover information and discern the tangled web.

4.1 Spatial Discovery of Servers

Today, CDNs and distributed cloud-based infrastructures are used to meet both scalability and reliability requirements, decoupling the owner of the content and the organization serving it. In this context some interesting questions arise: (*i*) Given a particular resource (i.e., a FQDN) what are all the servers or hosts that deliver the required content?, (*ii*) Do these servers belong to the same or different CDNs?, and (*iii*) Do CDNs catering to the resource change over time and geography? (*iv*) Are other resources belonging to the same organization served by the same or different set of CDNs?

DN-Hunter can easily answer all of the above questions. Algorithm 2 shows the pseudo-code for the Spatial Discovery functionality in DN-Hunter. The spatial discovery module first extracts the second-level domain name from the FQDN (line 4), and then queries the labeled flows database (line 5) to retrieve all $serverIP$ addresses in flows directed towards the second-level domain (i.e., the organization). Then, for every FQDN that belongs to the organization, the spatial discovery module will extract the $serverIP$ addresses that can serve the request (line 6-9) based on the DNS responses. This enables the module to: (*i*) Discover the information about the structure of servers (single server, or one/many CDNs) that handle all queries for the organization, (*ii*) Discover which servers handle a more specific resource. For example, different data centres/hosts may be serving the content for *mail.google.com* and *scholar.google.com*, and (*iii*)

[4]P2P hits are related to BitTorrent tracker traffic mainly.

```
SPATIAL DISCOVERY(FQDN)
Input: The targeted FQDN
Output: ranked list of serverIP addresses
2ndDomain ← FQDN.split()
ServerSet ←
    FlowDB.queryByDomainName(2ndDomain)
FQDNset ← 2ndDomain.query()
for all FQDN in FQDNSet do
    FQDN.ServerSet ←
        FlowDB.queryByDomainName(FQDN)
end for
Return(FQDN.ServerSet.sort(), ServerSet.sort())
```

Algorithm 2: Spatial Discovery Analytics Algorithm

Automatically keep track of any changes in *serverIP* addresses that satisfy a given FQDN over time. Note that the ability of DN-Hunter to easily track temporal and spatial changes in the FQDN-*serverIP* address mapping also enables some basic anomaly detection. While out of scope of this paper, consider the case of DNS cache poisoning where a response for certain FQDN suddenly changes and is different from what was seen by DN-Hunter in the the past. We can easily flag this scenario as an anomaly, enabling the security operator to take some action if required.

4.2 Content Discovery

As we saw in the previous subsection, a particular resource can be served by one or more CDNs or cloud infrastructures, and the spatial discovery analytics module provides deep insights into this. However, it is also important to understand tangle from another perspective. In other words, we need to answer the following questions: (*i*) Given a particular CDN what are the different resources that they host/serve? (*ii*) What is the popularity of particular CDNs in different geographies? (*iii*) Given two CDNs, what are the common resources that they both host?, and (*iv*) Does a given CDN focus on hosting content for certain types of services (like real-time multimedia streaming, mail, etc.)?

Once again DN-Hunter can answer the above questions easily based on the mapping stored in the flows database and using the whois database to associate IP addresses to CDNs. The complete algorithm for the content discovery module is shown in Algorithm 3. The algorithm takes a *ServerIPSet*, i.e., the set of *serverIP* addresses belonging to one or more CDNs, and extracts all the FQDNs associated with them (line 4-7). Depending on the desired granularity level, either the complete FQDN or only part of the FQDN (say, the second-level domain) can be considered. If only the second-level domains are considered, then the algorithm will return all the organizations served by the set of *serverIP* addresses provided as input. However, if only service tokens are used (we will discuss this in the next sub-section), then the algorithm will return which popular services are hosted by the input *serverIP* addresses.

4.3 Automatic Service Tag Extraction

Identifying all the services/applications running on a particular layer-4 port number is a legacy problem that network administrators encounter. Even today there are no existing solutions that can identify all applications on any given layer-4 port number. In fact, the network administrators depend on DPI solutions to address this problem. DPI technology can only provide a partial solution to this problem due to two reasons: (1) Several services/applications use encryption and hence bypass DPIs, and (2) DPI devices can only identify those services/applications for which they already have a signature, thus severely limiting the coverage.

```
1:  CONTENT DISCOVERY(ServerIPSet)
2:  Input: The list of targeted serverIP
3:  Output: The list of handled FQDNs
4:  DomainNameSet ← FlowDB.query(ServerIPSet)
5:  for all FQDN in DomainNameSet do
6:      TokenSet ← DomainName.split(FQDN)
7:  end for
8:  for all Token in TokenSet do
9:      Token.score.update()
10: end for
11: Return(Tokens.sort())
```

Algorithm 3: Content Discovery Analytics Algorithm

```
1:  TAG EXTRACTION(dPort, k)
2:  Input: targeted dPort, k of tags to return
3:  Output: The ranked list of tags
4:  DomainNameSet ← FlowDB.query(dPort)
5:  for all FQDN in DomainNameSet do
6:      TokenSet ← DomainName.split(NoTLD|NoSLD)
7:  end for
8:  for all Token in TokenSet do
9:      Token.score.update()
10: end for
11: Return(Tokens.sort(k))
```

Algorithm 4: Service Tag Extraction Analytics Algorithm

DN-Hunter provides a simple and automated way to address the above issue. The algorithm for extracting service tags on any layer-4 port number is shown in Algorithm 4. The input to the algorithm are the target port number and the k value for the top-k services to be identified. The algorithm first retrieves all FQDNs associated to flows that are directed to *dPort* (line 4). Each FQDN is then tokenized to extract all the sub-domains except for the TLD and second-level domain. The tokens are further split by considering non-alphanumeric characters as separators. Numbers are replaced by a generic N character (lines 5-7). For instance, *smtp2.mail.google.com* generates the list of tokens {smtpN, mail}.

We use the frequency of tokens as measure of "relevance" of the token for the targeted port (lines 8-10). To mitigate the bias due to some clients generating a lot of connections to a FQDN having the same token X, we use a logarithmic score. Mathematically, let $N_X(c)$ be the number of flows originated by *clientIP* c having the token X. Then the score of X is:

$$score(X) = \sum_c \log(N_X(c) + 1) \quad (1)$$

Tokens are then ranked by score and the top-k tokens are returned to the users (line 11). Depending on the final goal, different criteria can be applied to limit the list of returned tokens. For instance, the list can simply be limited to the top 5%, or to the subset that sums to the n-th percentile. Typically, the score distribution is very skewed, as we will show in Sec. 5.

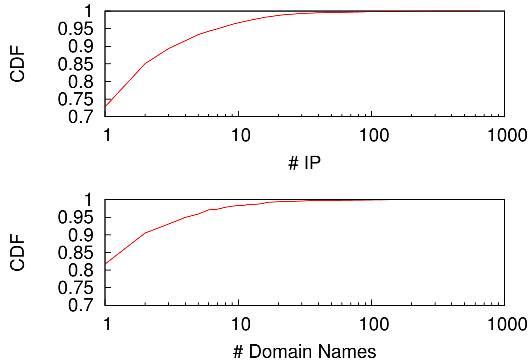

Figure 3: Number of $serverIP$ addresses associated to a FQDN (top) and number of FQDN associated to a $ServerIP$ (bottom). EU2-ADSL.

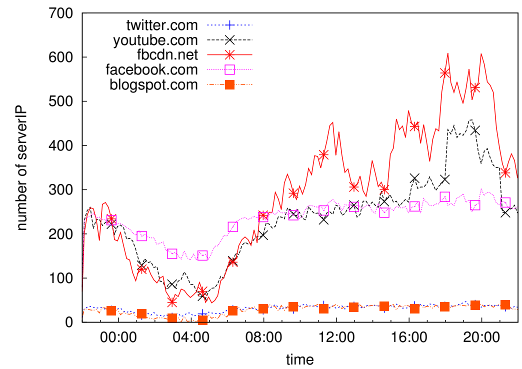

Figure 4: Number of IP addresses serving some particular 2nd-level domain name. EU1-ADSL2.

5. EXPERIMENTAL RESULTS

In this section, we present the results from using DN-Hunter on the traces mentioned in Sec. 2. We begin the discussion here by showing evidence of how tangled is the web today in terms of content, content providers, and hosts serving the content. We then present the results that clearly highlight the advantages of using DN-Hunter in an operational network compared to the existing solutions for traffic visibility and policy enforcement. In fact, DN-Hunter is now implemented as part of two different DPI tools and is deployed to provide traffic visibility to network operators. In the second half of this section we will present results from our advanced analytics modules to demonstrate the wide applicability and usefulness of DN-Hunter.

5.1 The Tangled Web

The basic hypothesis of this paper is that the web today is intertwined with content, content providers, and hosts serving the content that are continually changing over time and space. Hence we need a methodology that can assist in restoring clarity to operators regarding their network traffic. The top plot of Fig. 3 reports, for each FQDN, the overall number of $serverIP$ addresses that serve it. In the bottom plot of Fig. 3 we show the opposite - the number of different FQDNs a single $serverIP$ address serves. Fig. 3 was generated using the EU2-ADSL dataset, however, all the other datasets produced very similar result. We can clearly see that one single $serverIP$ is associated to a single FQDN for 73% of $serverIP$s, and 82% of FQDNs map to just one $serverIP$. But more important to note is that there are FQDNs that are served by hundreds of different $serverIP$ addresses. Similarly a large number of FQDNs are served by one $serverIP$. Notice the x-axis in this figure is presented in log scale.

Just looking at the one-to-many mapping between FQDN and $serverIP$ addresses reveals only a small part of the complexity. Now let us add time into the mix. Fig. 4 shows the number of $serverIP$ addresses that have been observed responding to some selected well-known second-level domains. We consider time bins of 10min, covering a 24h period from EU1-ADSL2 dataset. For some of the domains (like fbcdn.net and youtube.com) we can clearly see a diurnal pattern with more $serverIP$s being used during late evening when compared to early morning. In fact, for *youtube.com* we can see that there is a big and sudden jump in the number of $serverIP$s between 17:00 and 20:30. This reflects a change in the YouTube policies, triggered by the peak-time load. The domain *fbcdn.net* (owned by Akamai and serving Facebook static content) shows similar characteristics with more than 600 different $serverIP$ addresses serving content in every 10min interval between 18:00 and 20:00. Finally, some of the other domains like *blogspot.com* (aggregating more than 4,500 total FQDN) are served by less than 20 $serverIP$s even during peak traffic hours.

Fig. 5 reports the number of different FQDNs that were served every 10min by different CDNs and cloud providers over a period of 24h. The MaxMind organization database was used to associate $serverIP$ addresses to organization. We can clearly see that Amazon serves more than 600 distinct FQDN in every 10 min interval during peak hours (11:00 to 21:00). In total, Amazon served 7995 FQDNs in a day. While Akamai and Microsoft also serve significant number of FQDNs during peak hours, other CDNs like EdgeCast serve less than 20 FQDNs.

Another aspect worth noting here is that association between FQDNs and CDNs change over time and space (i.e., geography). Due to space constraints we do not present these results here. However, all of the above results clearly show why it is very hard to discern and control the traffic in today's networks! In fact, there is clear need for a solution like DN-Hunter that can track these changes seamlessly to ensure traffic visibility at any point in time. Surprisingly, the results presented in this section for motivating the need for a solution like DN-Hunter could not have been produced if we did not have DN-Hunter!

5.2 Traffic Visibility and Policy Enforcement

The key feature of DN-Hunter is to provide a "label" (i.e., the FQDN that the client was contacting) to every flow in the network automatically. To show how this labeling evolves over time, we show the results from our live deployment in EU1-ADSL2 for a period of 18 days in April, 2012. In Fig. 6 we report the total number of unique FQDNs over time. The plot shows the growth of unique entities - FQDNs, second-level domains, and $serverIP$ - over time. Once again we can clearly see the diurnal pattern where the increase in unique entities is much higher during the day than the night. After a steep growth during the first few days, the

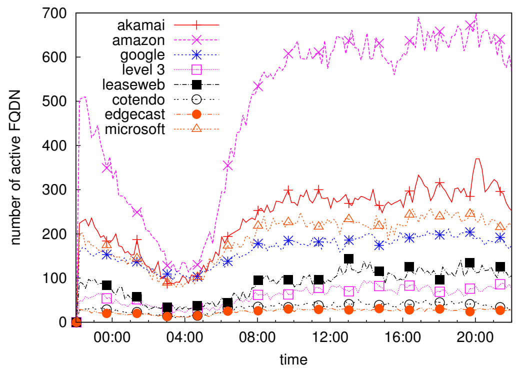

Figure 5: Number of FQDN served by CDNs through a day. EU1-ADSL2.

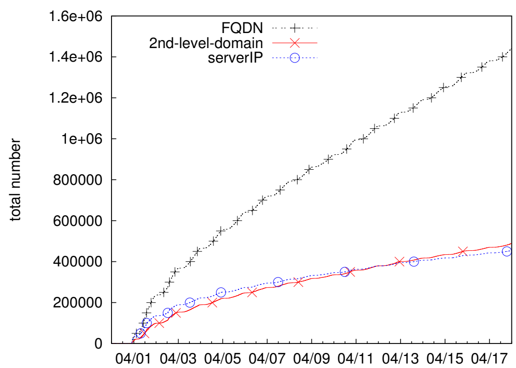

Figure 6: Unique FQDN, 2nd level domain names and IP birth processes. EU1-ADSL2 live.

number of unique $serverIP$ addresses and second-level domains reach a saturation point and do not grow much. This result basically indicates that the same $serverIP$ addresses are used to serve the contents for the same organizations (i.e., second-level domains). However, a surprising result is regarding the unique FQDNs. As we can see, the number of unique FQDNs keeps increasing even after 18 days of observation. In 18 days we saw more than 1.5M unique FQDNs and it was still growing at the rate of about 100K per day. This reflects the fact that the content being accessed on the Internet keeps growing, with new services popping up regularly. The main take away point is that in order to get fine-grained traffic visibility (and thus be applied for policy enforcement), it is critical to use a tool like DN-Hunter that can dynamically keep track of the content and their association with content providers and the hosts serving the content.

5.2.1 *The Case of Encrypted Traffic*

As we mentioned earlier, one of the main advantages of DN-Hunter when compared to traditional DPI solutions is its ability to label encrypted (TLS/SSL) flows. Traditional DPI solutions cannot identify encrypted traffic by inspecting the packet content and matching it against a signature. However, the DPI solution can be modified to inspect the certificates exchanged during the TLS/SSL handshake to figure out the server name of the organization that will provide the content.

Certificate equal FQDN	18%
Generic certificate	19%
Totally different certificate	40%
No certificate	23%

Table 4: Comparison between the server name extracted from TLS certificate-inspection and the FQDN using DN-Hunter. EU1-ADSL2.

In order to compare the certificate inspection approach with DN-Hunter, we implemented the certificate inspection functionality in Tstat. Tab. 4 compares certificate inspection approach with DN-Hunter for all TLS flows in the EU1-ADSL2 dataset. Results show that DN-Hunter clearly outperforms the certificate inspection approach. For 23% of the flows in the trace there was no certificate, while for 40% of the flows the server name in the certificate was totally different from the FQDN. For the other 37% of the flows that matched the second-level domain name in the FQDN, only 18% matched the complete FQDN. The main problems with the certificate inspection approach are three-fold: (*i*) The server name can be "generic", e.g., $*$.*google.com*, thus not giving the fine-grained visibility into the actual services. (*ii*) The server name may indicate the server used by the hosting CDN and may not reflect anything about the service, e.g., *a248.akamai.net* in the certificate for providing Zynga content, and (*iii*) Certificate exchange might happen only the first time a TLS/SSL server is contacted and all other flows following that will share the trust. Thus using such an approach is almost infeasible.

5.3 Spatial Discovery of Servers

The main goal of the spatial discovery module is to track a particular resource (FQDN or second-level domain) to understand which $serverIP$s and CDNs serve the requested content. For the ease of exposition, in this section, we will focus on two specific second-level domains - LinkedIn and Zynga. Fig. 7 shows the mapping between the various FQDNs of LinkedIn and the CDNs serving the content in US-3G dataset. The oval nodes represent DNS tokens extracted from the FQDNs, while arcs connect the tokens to reconstruct the FQDN. The numbers in these tokens are represented as a generic letter, N. The rectangular nodes group tokens by the CDN hosting them based on the information from the MaxMind database. To illustrate the concept better let us consider the leftmost branch in Fig. 7. The complete FQDN is the concatenation of all the tokens, i.e., *mediaN.linkedin.com*. These FQDNs are served by Akamai CDN using 2 servers and accounts for 17% of the total flows destined to *linkedin.com*. In order to limit the size of the figure, we have hidden 7 different tokens in the rightmost branch of the tree.

From the figure, it is easy to see that LinkedIn relies on the service offered by several CDN providers. Only the *www.linkedin.com* FQDN along with 7 other FQDNs are served by Linkedin managed servers. Most of the static content is served by hosts in three different CDNs - Akamai, CDNetwork, and Edgecast. In fact, EdgeCast serves 59% of all flows with a single $serverIP$ address. On the

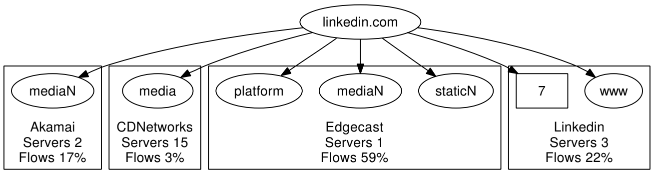

Figure 7: Linkedin.com domain structure served by two CDNs. US-3G.

contrary, CDNetworks, serves only 3% of flows with 15 different $serverIP$ addresses.

Let us now consider the second sample domain - Zynga (see Fig. 8). We can see that Amazon EC2 cloud service provides computational resources required for the games while Akamai CDN hosts most of the static content. Some services/games like MafiaWars are served directly by Zynga owned servers. Interestingly, about 500 Amazon $serverIP$ addresses are contacted and they handle 86% of all Zynga flows. Akamai serves fewer requests (7%); yet, 30 different $serverIP$ are observed.

Given that the off-line analyzer relies on actual measurement of network traffic, it is able to capture both the service popularity among the monitored customers, and the bias induced by the server selection and load balancing mechanisms. To elaborate this further, let us consider Fig. 9. Each of the three sub-figures corresponds to a different content provider (i.e., the second-level domain name). For each of these content providers we plot the access patterns in three of our traces (EU1-ADSL1, US-3G, and EU2-ADSL). In other words, the x-axis in each of these graphs are the CDNs hosting the content and the y-axis represents different traces. Notice that for every CDN, the figure shows all the accessed $serverIP$ addresses that belong to the CDN. Hence the width of the column representing each CDN is different. Also, the gray scale of each block in the graph represents the frequency of access; the darker is a cell, the larger is the fraction of flows that a particular $serverIP$ was responsible for. The "SELF" column reports cases in which the content providers and content hosts are the same organization.

The top graph in Fig. 9 shows the access pattern for Facebook. We can see that in all the datasets, most of the Facebook content is hosted on Facebook servers. The only other CDN used by Facebook is Akamai, which uses different $serverIP$ in different geographical regions. In the middle graph, we can see that Twitter access patterns are a little different. Although, Twitter relies heavily on its own servers to host content, they also rely heavily on Akamai to serve content to users in Europe. However, the dependence on Akamai is significantly less the US. The bottom graph shows the access patterns for Dailymotion, a video streaming site. Dailymotion heavily relies on Dedibox to host content both in Europe and US. While they do not host any content in their own servers in Europe, they do serve some content in the US. Also, in the US they rely on other CDNs like Meta and NTT to serve content while they rely a little bit on Edgecast in Europe.

5.4 Content Discovery

Although the spatial discovery module provides invaluable insight into how a particular resource is hosted on various

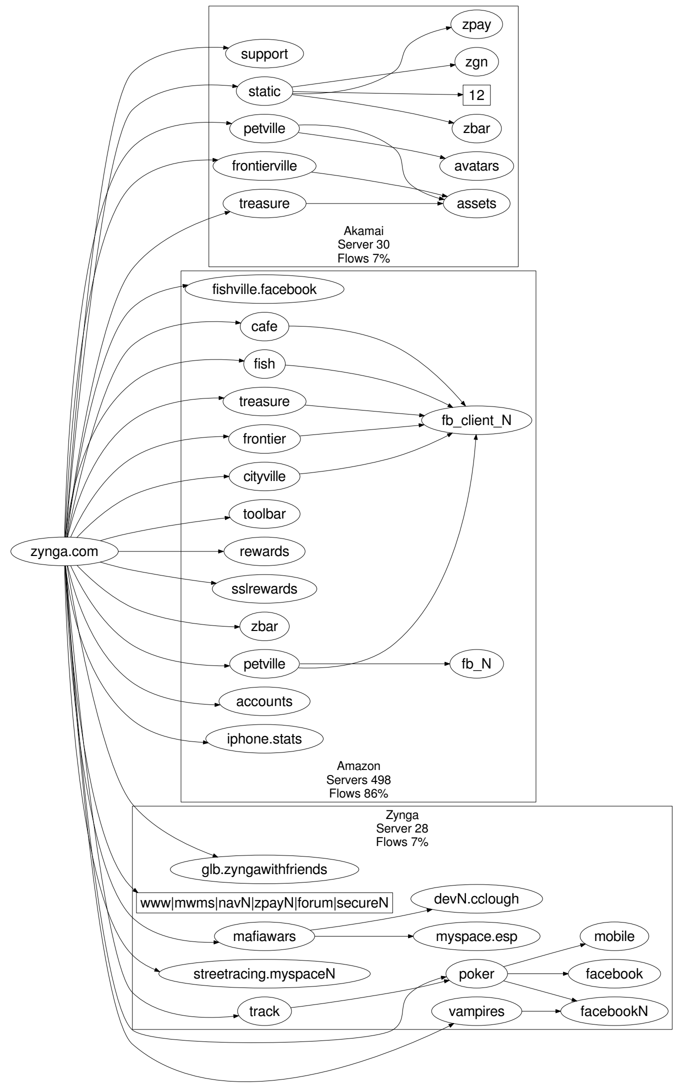

Figure 8: Zynga.com domain structure served by two CDNs. US-3G.

CDNs, it does not help in understanding the complete behavior of CDNs. In the content discovery module our goal is to understand the content distribution from the perspective of CDNs and cloud service providers. Tab. 5 shows the top-10 second-level domains served by the Amazon EC2 cloud in EU1-ADSL1and US-3G. Notice that one dataset is from Europe and the other from US. We can clearly see that the top-10 in the two datasets do not match. In fact, some of the popular domains hosted on Amazon for US users like admarvel, mobclix, and andomedia are not accessed on Amazon by European users, while other domains like cloutfront, invitemedia, and rubiconproject are popular in both the datasets. This clearly shows that the popularity and access patterns of CDNs hosting content for different domains depend on geography; extrapolating results from one geography to another might result in incorrect conclusions.

5.5 Automatic Service Tag Extraction

Another interesting application of DN-Hunter is in iden-

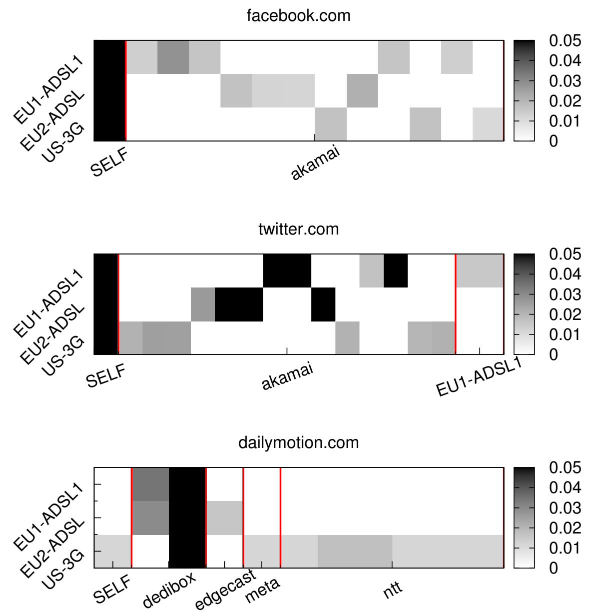

Figure 9: Organizations served by several CDN according to viewpoint.

Rank	US-3G	%	EU1-ADSL1	%
1	cloudfront.net	10	cloudfront.net	20
2	invitemedia.com	10	playfish.com	16
3	amazon.com	7	sharethis.com	5
4	rubiconproject.com	7	twimg.com	4
5	andomedia.com	5	amazonaws.com	4
6	sharethis.com	5	zynga.com	4
7	mobclix.com	4	invitemedia.com	2
8	zynga.com	3	rubiconproject.com	2
9	admarvel.com	3	amazon.com	2
10	amazonaws.com	3	imdb.com	1

Table 5: Top-10 domains hosted on the Amazon EC2 cloud.

tifying all the services/applications running on a particular layer-4 port number. This application is only feasible due to the fined grained traffic visibility provided by DN-Hunter. To keep the tables small, we only show the results extracted on a few selected layer-4 ports for two data sets - EU1-FTTH (Tab. 6) and US-3G(Tab. 7). In these tables we show the list of terms along with the weights returned by the Service Tag Extraction Analytics algorithm (Algorithm 4). The last column in each of these tables is the ground truth obtained using Tstat DPI and augmented by Google searches and our domain knowledge.

We can clearly see that the most popular terms extracted in both the datasets in fact represents the application/service on the port. Some of them like pop3, imap, and smtp are very obvious by looking at the top keyword. However, some of the other are not very obvious, but can be derived very easily. For example, consider the port 1337. TCP port 1337 is not a standard port for any service and even a google search for TCP port 1337 does not yield straight forward results. However by adding "exodus" and "genesis", the main keywords extracted in DN-Hunter, to the google search along

Port	Keywords	GT
25	(91)smtp, (37)mail, (22)mxN, (19)mailN, (18)com, (17)altn, (14)mailin, (13)aspmx, (13)gmail	SMTP
110	(240)pop, (151)mail, (68)popM, (33)mailbus	POP3
143	(25)imap, (22)mail, (12)pop, (3)apple	IMAP
554	(1)streaming	RTSP
587	(10)smtp, (3)pop, (1)imap	SMTP
995	(101)pop, (37)popN, (31)mail, (20)glbdns (20)hot, (17)pec	POP3S
1863	(21)messenger, (5)relay, (5)edge, (5)voice, (2)msn, (2)com, (2)emea	MSN

Table 6: Keyword extraction example considering well-known ports. EU1-FTTH.

Port	Keywords	GT
1080	(51)opera, (51)miniN	Opera Browser
1337	(83)exodus, (41)genesis	BT Tracker
2710	(62)tracker, (9)www	BT Tracker
5050	(137)msg, (137)webcs, (58)sip, (43)voipa	Yahoo Messager
5190	(27)americaonline	AOL ICQ
5222	(1170)chat	Gtalk
5223	(191)courier, (191)push	Apple push services
5228	(15022)mtalk	Android Market
6969	(88)tracker, (19)trackerN, (11)torrent, (10)exodus	BT Tracker
12043	(32)simN, (32)agni	Second Life
12046	(20)simN, (20)agni	Second Life
18182	(92)useful, (88)broker	BT Tracker

Table 7: Keyword Extraction for frequently used ports; Well-known ports are omitted. US-3G.

with TCP port 1337 immediately shows that this port in US-3G dataset is related to www.1337x.org BitTorrent tracker.

5.6 Case Study - appspot.com Tracking

In this section, we want to present a surprising phenomenon that we discovered using DN-Hunter's ability to track domains. Let us consider the domain *appspot.com*. *Appspot* is a free web-apps hosting service provided by Google. The number of applications, CPU time and server bandwidth that can be used for free are limited. Using the labels for various flows in the labeled flows database, we extract all traffic associated with services. This allows to understand the kind of applications and services that are hosted in the Appspot cloud.

Fig. 10 shows the most relevant applications hosted on *appspot* as a word cloud where the larger/darker fonts represent more popular applications. Although *appspot* is intended to host legacy applications, it is easy to see that users host applications like "open-tracker", "rlskingbt", and the like. A little more investigation reveals that these applications actually host *BitTorrent* trackers for free. With the help of the information from DN-Hunter and also the Tstat DPI deployed at the European ISP, we find that there are several trackers and other legacy applications running in the *appspot.com* site. We present the findings in Tab. 8. As we can see, BitTorrent trackers only represent 7% of the applications but constitute for more flows than the other applications. Also, when considering the total bytes exchanged for each of these services, the traffic from client-to-server gen-

Figure 10: Cloud tag of services offered by Google Appspot. EU1-ADSL2 during live deployment.

Service Type	Services	Flows	C2S	S2C
Bittorrent Trackers	56	186K	202MB	370MB
General Services	824	77K	320MB	5GB

Table 8: Appspot services. EU1-ADSL2 live.

erated by the trackers is a significantly large percentage of the overall traffic.

In Fig. 11 we plot the timeline (in 4hr intervals) of when the trackers were active over a period of 18 days. A dot represents that the tracker was active at that time interval. We assign each tracker an id, starting at 1 and incrementally increasing based on the time when the tracker was first observed. Of all the 45 trackers seen in the 18 day period, about 33% (colored in red, ids 1-15) of them remained active for all the 18 days. Trackers with ids 26-31 (colored in blue) exhibit a unique pattern of on-off periods. In other words, all of these trackers were accessed during the same time intervals. Such a synchronized behavior indicates, with high probability, that one BitTorrent client was part of a swarm, so that when the client was active, the tracker was accessed. Similar considerations hold for the trackers with ids 33-34 (colored in green), which was seen accessed on May 5th for the first time.

Finally, trackers with ids larger than 33 highlight a birth process according to which a new tracker is born and accessed by only a few BitTorrent clients. This is due to the particular environment the trackers live: since Appspot limits the amount of resource an application can use for free, new trackers are born once they run out to resources. Indeed, checking the status of the trackers during May 15th 2012, we verified that most of them, while still existing as FQDN, run out of resources and made unavailable by Google. They live as *zombies*, and some BitTorrent clients are still trying to access them.

6. DIMENSIONING THE FQDN CLIST

In Sec. 3, we presented the design of the DNS resolver. One of the key data structures of the DNS resolver is the FQDN Clist. Choosing the correct size for the Clist is critical to the success of DN-Hunter. In this section we will present a methodology to choose the correct value of L (size of the Clist) and the real-time constraint implication.

Fig. 12 shows the Cumulative Distribution Function (CDF) of the "first flow delay", i.e., the time elapsed between the observation of the DNS response directed to $clientIP$ and the first packet of the *first* flow directed to one of the $serverIP$ addresses in the answer list. Semilog scale is used for the sake of clarity. In all datasets, the first TCP flow is observed

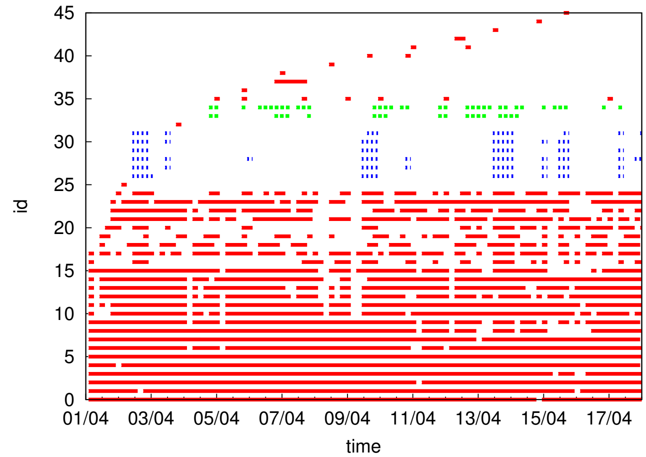

Figure 11: Temporal evolution of the BitTorrent trackers running on Appspot.com. EU1-ADSL2 during live deployment.

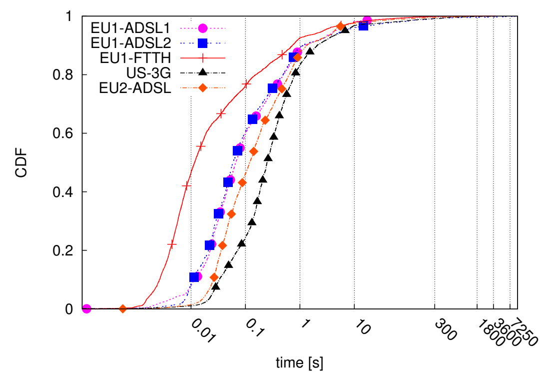

Figure 12: Time elapsed between DNS response and the *first* TCP flow associated to it.

after less than 1s in about 90% of cases. Access technology and sniffer placement impact this measurement; for instance, FTTH exhibits smaller delays, while the 3G technology suffers the largest values.

Interestingly, in all traces, for about 5% of cases the first flow delay is higher than 10s, with some cases larger than 300s. This is usually a result of aggressive pre-fetching performed by applications (e.g., web browsers) that resolve all FQDNs found in the HTML content before a new resource is actually accessed. Tab. 9 quantifies the fraction of "useless" DNS responses, i.e., DNS queries that were not followed by any TCP flow. Surprisingly, about half of DNS resolutions are useless. Mobile terminals are less aggressive thus resulting in lower percentage of useless responses.

Trace	Useless DNS
EU1-ADSL1	46%
EU1-ADSL2	47%
EU1-FTTH	50%
EU2-ADSL	47%
US-3G	30%

Table 9: Fraction of useless DNS resolution.

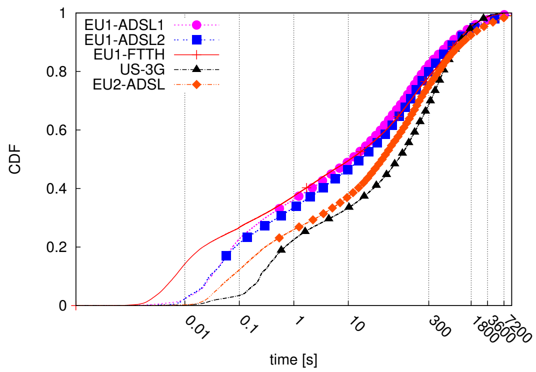

Figure 13: Time elapsed between a DNS response and *any* TCP flow associated to it.

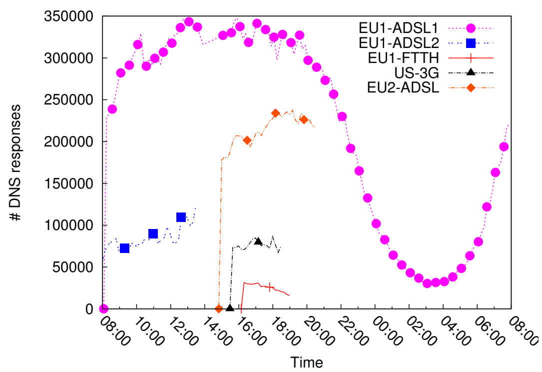

Figure 14: DNS responses observed during a day by intervals of 10 minutes in EU1-ADSL1.

Fig. 13 shows the CDF of the time elapsed between the DNS response and *any* subsequent TCP flow the client establishes to any of the *serverIP* addresses that appeared in the answer list. It reflects the impact of caching lifetime at the local DNS resolver at clients. The initial part of the CDF is strictly related to the first flow delay (Fig. 12); subsequent flows directed to the same FQDN exhibit larger time gaps. Results show that the local resolver caching lifetime can be up to few hours. For instance, to resolve about 98% of flows for which a DNS request is seen, *Clist* must handle an equivalent caching time of about 1 hour.

Fig. 14 shows the total number of DNS responses observed in 10m time bins. As we can see, at the peak time about 350,000 requests in EU1-ADSL1 dataset. In this scenario, considering a desired caching time of 1h, L should be about 2.1M entries to guarantee that the DNS resolver has an efficiency of 98%.

We have also checked the number of *serverIP* addresses returned in each DNS response. Since the *clientIP* can choose any one of the *serverIP* addresses to open the data connection, all of the *serverIP* addresses must be stored in the DNS resolver. The results from all the datasets are very similar with about 40% of responses returning more than one *serverIP* address. About 20-25% of responses include 2-10 different ip-addresses. Most of these are related to servers managed by large CDNs and organizations. For example, up to 16 *serverIP*s are returned when querying any Google FQDN. The maximum number exceeds 30 in very few cases.

Trace	Total Test	Generic coll.	Severe Coll
US-3G	1.3M	0.7%	0.6%
EU2-ADSL	11.9M	0.7%	0.3%
EU1-ADSL1	4.9M	0.01%	0.004%
EU1-ADSL2	2.6M	0.1%	0.05%
EU1-FTTH	680k	1.6%	0.9%

Table 10: DN-Hunter Collision Ratio

6.1 Collision probability in practice

Now we consider the possibility of collisions when the same *clientIP* is accessing two or more FQDNs hosted at the same *serverIP*. DN-Hunter would return the last observed FQDN, thus possibly returning incorrect labels. We call this event a "collision". We examined the traces to see how frequently such situation occurs. We observed was that the most common reason for this is due to HTTP redirection, e.g., *google.com* being redirected to *www.google.com* and then to *www.google.it*, all FQDNs being served by the same *serverIP*. Hence, two types of collisions are possible: (*i*) generic collision, and (*ii*) severe collision. The latter accounts for the cases in which the two conflicting FQDNs have different second level domain names, e.g., *google.com* and *youtube.com*.

Collisions can occur in DN-Hunter when the same *clientIP* address contacts the same *serverIP* address several times during the period of our traces. In the second column of Tab. 10 we report the total number of flows that are potential candidates for collision in DN-Hunter. If the domain name that DN-Hunter return for the two successive flow from the same <*clientIP*, *serverIP*> pair is the same, then we declare that there is no collision. If the returned domain name is different, then we declare that a collision has occurred. If a collision occurs, then we examine the domain names to determine if the collision is a generic one or a severe one. The second column of Tab. 10 reports the total number of potential candidate flows for collision. The third and the fourth columns report the percentage of the potential collision candidates that translate into generic and severe collisions respectively. As we can clearly see, the collision rate is extremely low in all the traces used for evaluation. In fact, we observe that more than 70% of collisions occur within a 1 second interval, suggesting that these collisions could be a result of HTTP redirection. Although, these are collisions might not impact the accuracy of DN-Hunter, in Tab. 10 we still consider them to be collisions and give a conservative estimate. In summary, the problem of collisions has minimal impact on the operational value of DN-Hunter.

Finally, note that it is possible to extend DN-Hunter to return all possible labels associated to a flow instead of only the latest one, thus giving the network administrator the ability to resolve the collisions using more advanced policies than by strictly using the latest FQDN.

6.2 Deployment issue

DN-Hunter is a passive sniffer which assumes to observe DNS and data traffic generated by the end-users. The natural placement of the sniffer is at the network boundaries, where end-users' traffic can be observed. The Flow Sniffer and the DNS Response Sniffer may also be placed at different vantage points, e.g., the latter may be located in front of (or integrated into) the internal DNS server to intercept all DNS queries. Considering DNS traffic sniffing, DNSSEC [3] poses no challenge since it does not provide confidentiality to DNS traffic. DNSCrypt [4], a recent proposal to encrypt DNS traffic, on the contrary, would make the DNS Response Sniffer ineffective. DNSCrypt is not yet widely deployed and it requires significant DNS infrastructure [4] changes to be pragmatic in the near future [5].

DN-Hunter labels TCP flows using $(clientIP, serverIP)$ pair as the key. In scenarios where the same $clientIP$ is shared by multiple end-hosts via NAT/connection sharing tools, DN-Hunter may associate the wrong FQDN to the flow,causing a possible collision. Notice that this can happen only if the end-hosts are initiating connection at almost the same time to the same $serverIP$ (which has been returned when requesting two different FQDN). This can also happen when a end-user is using multiple applications (e.g., multiple browsers windows/tabs opened at the same time) to access different services running on the same $serverIP$ at the same time. We cannot quantify how probable these events are in practice in our traces, and given the very low collision probability (see Sec. 6.1), we expect this to be a marginal problem in the considered scenarios. We argue that in access scenarios where single households are allowed to share the internet connection among few terminals, the collision probability is still marginal. However, in the scenario where there are several hundreds (or thousands) of users behind NAT and the monitoring point for DN-Hunter is after the NAT, DN-Hunter will be severely impaired. Deploying DN-Hunter before the NAT will result in accurate results.

7. RELATED WORK

DNS has been a popular area of research over the past few years. I this section we will highlight the main differences between DN-Hunter and some of the other related works.

The first set of related work focusses on exploring the relationship between CDNs and DNS mainly to study the performance and load balancing strategies in CDNs [6–8]. Ager et al. [9] complement this by proposing an automatic classifier for different types of content hosting and delivery infrastructures. DNS traces actively collected and provided by volunteers are analyzed, in an effort to provide a comprehensive map of the whole Internet. DN-Hunter is similar in spirit, but leverages DNS information in a completely passive manner and focuses on a broader set of analytics.

Similar to our work, [10, 11] focus on the relationship between FQDNs and the applications generating them mainly in the context of botnet detection. However, in DN-Hunter, we mainly focus on identifying and labeling various applications in the Internet. Furthermore, we focus on some advanced analytics to shed light on problems that are critical for untangling the web.

[12] analyze the DNS structure using available DNS information on the wire. The authors define 3 classes of DNS traffic (canonical, overloaded and unwanted), and use the "TreeTop" algorithm to analyze and visualize them in real-time, resulting in a hierarchical representation of IP traffic fractions at different levels of domain names. DN-Hunter goes beyond the visualization of DNS traffic as the set of domain names being used by users in a network, and provides a much richer information to understand today's Internet.

The same authors above extend their analysis on DNS traffic in [13]. Their proposal is similar to the DN-Hunter Sniffer goal, even if not designed to work in real time: flows are labeled with the original resource name derived from the DNS (as in the Flow Database). Then, flows are classified in categories based on the labels of the DNS associated entries. This allows to recover the "volume" of traffic, e.g., going to .com domain, or to apple.com, etc. Authors then focus on the study of breakdown of traffic volumes based on DNS label categories. As presented in the paper, DN-Hunter Analyzer performs much more advanced information recovery out of DNS traffic.

In [14], the authors focus on security issues related to DNS prefetching performed by modern Internet browsers, specifically the fact that someone inspecting DNS traffic can eventually reconstruct the search phrases users input in the search boxes of the browser. Their methodology is somewhat similar to the one DN-Hunter uses to associate tags to network ports, but the objective is completely different.

Note CDN content discovery and mapping the geographical locations of servers is a topic that could have been addressed without the introduced approach. Yet, the major plus of DN-Hunter is that it is completely passive, and very simple. It provides information to the ISP that naturally reflects its end-users habits along with the configuration CDNs adopt for serving traffic generated by ISP's end-users. In other words, it reflects the "current" usage of the network with respect to the traffic and external events (like CDN policy changes).

8. CONCLUSIONS

In this work we have introduced DN-Hunter, a novel tool that links the information found in DNS responses to traffic flows generated during normal Internet usage. Explicitly aimed at discerning the tangle between the content, content providers, and content hosts (CDNs and cloud providers), DN-Hunter unveils how the DNS information can be used to paint a very clear picture, providing invaluable information to network/security operators. In this work, we presented several applications of DN-Hunter, ranging from automated network service classification to dissecting content delivery infrastructures.

As notable examples, we have shown how DN-Hunter is helpful (*i*) in providing a fine-grained traffic visibility even when the traffic is encrypted (i.e., TLS/SSL flows), (*ii*) in identifying flows even before the flows begin, thus providing superior network management capabilities to administrators, (*iii*) in observing and tracking how CDNs and cloud providers host content for a particular resource over time, and (*iv*) in discerning the services/contents hosted by a CDN or cloud provider. We believe that the applications of DN-Hunter the Analyzer in particular, are not limited to the ones presented in this work, and novel applications can leverage the information exposed by the labeled flows database.

DN-Hunter has been deployed in actual ISP networks, providing network administrators enhanced traffic visibility and the ability to perform detailed forensics if required. A key challenge for the future is to devise automatic algorithms to mine the amount of data exposed by DN-Hunter and by network monitoring tools in general. We believe that this challenge is common to the network monitoring and measurement community, and modern tools have to be devised to dig into this enormous volume of data.

Acknowledgments

The research leading to these results has received funding from Narus Inc. and from the European Commission under the Seventh Framework Programme (FP7 2007-2013) with the Grant Agreement n. 318627 - Integrated Project "mPlane - an Intelligent Measurement Plane for Future Network and Application Management".

9. REFERENCES

[1] V. Gehlen, A. Finamore, M. Mellia, and M. Munafò. Uncovering the big players of the web. In *TMA Workshop*, pages 15–28, Vienna, AT, 2012.

[2] A. Finamore, M. Mellia, M. Meo, M.M. Munafo, and D. Rossi. Experiences of internet traffic monitoring with tstat. *Network, IEEE*, Vol.25, N.3, pages 8–14, May 2011.

[3] R. Arends et. Al. RFC 4033 - DNS Security Introduction and Requirements, March 2005.

[4] Introducing DNSCrypt (Preview Release), February 2011. `http://www.opendns.com/technology/dnscrypt/`

[5] B. Ager, H. Dreger, and A. Feldmann. Predicting the DNSSEC Overhead using DNS Traces. In *40th Annual Conference on Information Sciences and Systems*, pages 1484–1489. IEEE, 2006.

[6] S. Triukose, Z. Wen, and M. Rabinovich. Measuring a Commercial Content Delivery Network. In *ACM WWW*, pages 467–476., Hyderabad, IN, 2011.

[7] A.J. Su, D.R. Choffnes, A. Kuzmanovic, and F.E. Bustamante. Drafting Behind Akamai: Inferring Network Conditions Based on CDN Redirections. *IEEE/ACM Transactions on Networking*, Vol.17, N. 6, pages 1752–1765, 2009.

[8] C. Huang, A. Wang, J. Li, and K.W. Ross. Measuring and Evaluating Large-scale CDNs. In *ACM IMC*, pages 15–29, Vouliagmeni, GR, 2008.

[9] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Web Content Cartography. *ACM IMC*, pages 585–600, Berlin, DE, 2011.

[10] H. Choi, H. Lee, H. Lee, and H. Kim. Botnet Detection by Monitoring Group Activities in DNS Traffic. In *IEEE CIT* , pages 715–720., Fukushima, JP, 2007.

[11] S. Yadav, A.K.K. Reddy, AL Reddy, and S. Ranjan. Detecting Algorithmically Generated Malicious Domain Names. In *ACM IMC*, pages 48–61., Melbourne, AU, 2010.

[12] D. Plonka and P. Barford. Context-aware Clustering of DNS Query Traffic. In *ACM IMC*, pages 217–230., Vouliagmeni, GR, 2008.

[13] D. Plonka and P. Barford. Flexible Traffic and Host Profiling via DNS Rendezvous. In *Workshop SATIN*, 2011.

[14] S. Krishnan and F. Monrose. An Empirical Study of the Performance, Security and Privacy Implications of Domain Name Prefetching. In *IEEE/IFIP DSN*, Boston, MA, 2011.