Technical Report

UCAM-CL-TR-748
ISSN 1476-2986

Number 748

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

GTVS: boosting the collection of application traffic ground truth

Marco Canini, Wei Li, Andrew W. Moore

April 2009

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

© 2009 Marco Canini, Wei Li, Andrew W. Moore

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

GTVS: boosting the collection of application traffic ground truth

Marco Canini, Wei Li, Andrew W. Moore

Abstract

Interesting research in the areas of traffic classification, network monitoring, and application-orient analysis can not proceed with real trace data labeled with actual application information. However, hand-labeled traces are an extremely valuable but scarce resource in the traffic monitoring and analysis community, as a result of both privacy concerns and technical difficulties: hardly any possibility exists for payloaded data to be released to the public, while the intensive labor required for getting the ground-truth application information from the data severely constrains the feasibility of releasing anonymized versions of hand-labeled payloaded data.

The usual way to obtain the ground truth is fragile, inefficient and not directly comparable from one's work to another. This chapter proposes and details a methodology that significantly boosts the efficiency in compiling the application traffic ground truth. In contrast with other existing work, our approach maintains the high certainty as in hand-verification, while striving to save time and labor required for that. Further, it is implemented as an easy hands-on tool suite which is now freely available to the public.

In this paper we present a case study using a 30 minute real data trace to guide the readers through our ground-truth classification process. We also present a method, which is an extension of GTVS that efficiently classifies HTTP traffic by its purpose.

1 Introduction

Ground truth[1] network information is a fundamental necessity to provide trustworthy results for accounting, modeling, measurement and performance evaluation purposes.

Further, high-confidence ground-truth data is fundamental to evaluate new methods for traffic classification and new uses of classification systems.

The lack of ground truth has adversely affected the soundness of many previous works [1]. For example, a lot of papers were evaluating and comparing application identification methods without accurate ground truth (i.e., relied just on port numbers) or using traces having a significant amount of unknown traffic.

The collection of ground-truth application information of the Internet traffic is critical to both research community and the industry:

[1]The certain knowledge of the associations between each traffic flow and the application causing it.

- it is the basis and a critical procedure to build applications for information assurance, network monitoring and traffic accounting, and the only way to evaluate their accuracy,
- it facilitates the networking research on nearly every aspect related to applications, protocols, network modeling and data analysis, so that methodologies and models can be derived and updated, and
- it provides fundamental, accurate knowledge of how people use the network; such knowledge now has increasing importance for network security and enterprise network management.

However, because of significant privacy concerns, hardly any payloaded data can be released, while publicly accessible anonymized non-payloaded traces (e.g., LBNL [2], CAIDA [3] and MAWI [4]) are of limited value without accurate application information associated with them. A common practice becomes to hand-verify the ground truth from payloaded trace collected on one's own or friendly links.

Obtaining the application ground truth from data traces is difficult. On one hand, there are too much data to be resolved before it is possible to build a sufficient knowledge base for a desired task. On the other hand, there is too few information, much of which could even be deceiving, to obtain the knowledge from.

Many different schemes have been used in the past to obtain the ground truth for studies: Moore and Papagiannaki [5] documented a fine-grained classification scheme comprising nine identification methods. The ground-truth labels used in [6] were based on "hand-classification", while the authors of [7] and [8] were using an "automated payload-based classification" as they described. The collection of many (if not all) of these ground-truth data was automated using extensible NIDSes (Network Intrusion Detection Systems) such as Snort [9] and Bro [10], or through homemade scripts. The efforts made to collect the ground-truth data were both significant and highly improvised, causing a lot of unnecessary, repeated labor, untrustworthy results (e.g., ground truth derived by signature matching alone) and inconsistency (e.g., different levels of confidence and completeness) between different works. Further, there is often a lack of verification mechanisms between multiple information sources, hence faults are inevitable and unrecoverable.

As a consequence of the inconsistency of the analyzed data sets, it is not possible to rigorously and systematically compare the results of the various classification methods. This also means that the operational community cannot decide on which method to use and when.

In this paper, we present GTVS (Ground Truth Verification System), a novel framework and methodology dedicated to ground-truth verification. It automates the data manipulation and information retrieval processes as well as significantly increasing the efficiency of human hand-verification methodology. It works at a finest granularity of a bi-directional flow defined by the standard IP 5-tuple: {src IP, src port, dst IP, dst port and proto.}. It provides aggregated views and suggests heuristic rules based on multiple criteria, allowing for better recognition of the host behaviors and acceleration of the verification. Finally, it is extensible to allow additional sources of information, user-defined heuristics and different classifications, and to achieve different goals.

GTVS boosts the collection of application ground truth by significantly reducing the time and labor required in the process; it preserves and improves the general quality of the ground truth and facilitates validations among many information sources. Finally, it provides a neat and handy platform to facilitate the management of hand-verification projects and to allow experiences and data to be shared among the research community.

The following Section reviews and validates an important assumption used in our system. Section 3 presents an overview of the GTVS framework. Then, a detailed case study on a 30 minute real data trace is presented to guide the readers through our ground-truth classification process in Section 4. Section 5 details a complementary methodology targeted specifically at HTTP traffic and applies it to highlight a few characteristics of how the HTTP traffic has evolved in recent years. Related works are discussed in Section 6 and Section 7 concludes the paper.

2 Assumption and Validation

We observe that flows belonging to the same service or application often share a subset of the IP 5-tuple, notably, the {dst IP, dst port, proto.} and {src IP, dst IP} sub-tuples. For example, TCP packets to and from 207.46.107.73:1863 would be associated to the Windows Live Messenger.

This leads to an assumption that underpins our approach: *flows of the same sub-tuples are associated to the same service or application.* With this, the data can be labeled at high-level aggregations to accelerate the process. Similar assumptions are implicitly used in BLINC [7] where traffic is classified using "graphlets" which essentially resolve into sub-tuples.

We base our assumption on observations of institutional networks (for several institutions) taken over multi-day periods covering periods since 2003 up until the current day. Within each day, we identify flows through pattern matching and manual inspection of the payload contents. The results show that traffic follows our stability assumption for the entire day in most cases with few exceptions as separately discussed below.

The {dst IP, dst port} sub-tuple. Exceptions are different application encapsulated in VPNs and SOCKS proxies. Currently, in our settings, this traffic is categorized into "remote access" class. Others have discussed further mechanisms that can be applied to identify the encapsulated applications as needed [11]. We also point out that HTTP represents an exception whenever the protocol is used as a generic transport protocol. However, once the traffic is identified as HTTP by means of our assumption, further refinements can be applied as later described in section 5.

The {src IP, dst IP} sub-tuple. Exceptions include VPNs and SOCKS proxies as well as circumstances where there are multiple applications between a server and a client, e.g., a server operating both SSH and Web service. However, in such circumstances, the server is usually operating traditional services (e.g., Web, FTP, mail or SSH). This sub-tuple effectively complements the one above in classifying applications on multiple ports (e.g., FTP transfers, P2P and media streaming with port hopping).

The consistency assumption is implicit throughout the design and use of GTVS.

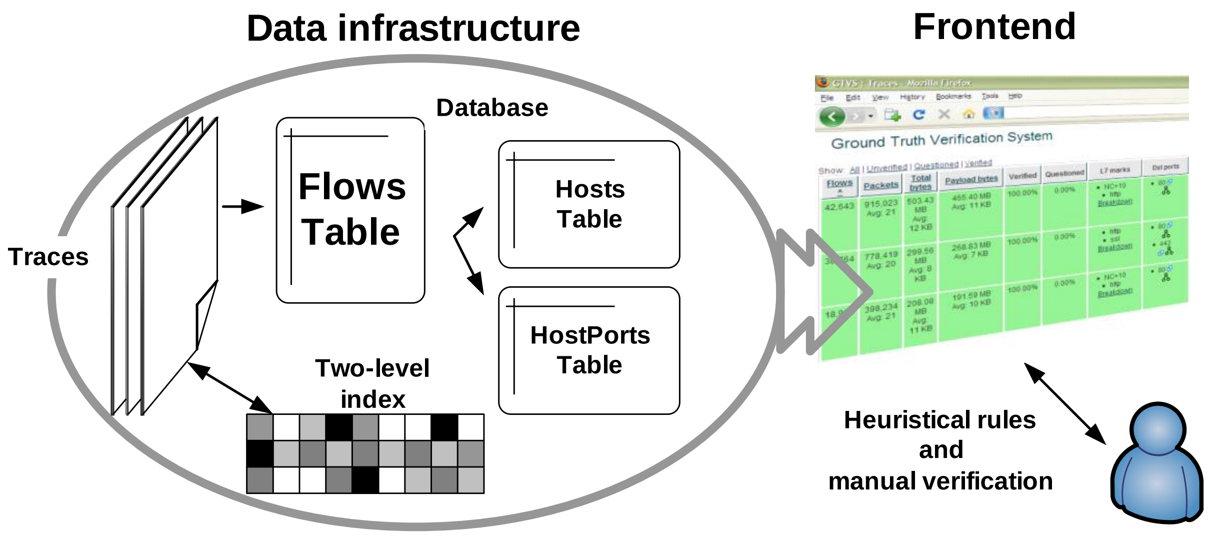

Figure 1: GTVS structure overview.

3 Overview

GTVS can be described in a layered structure, supporting a user-oriented design, as shown in Figure 1. It supports the work-flow a user would follow when manually verifying the flows, and collects almost all information that one would maximally find by hand-classification. It is composed of (*i*) a basic infrastructure layer for data management including packet traces and flows database, (*ii*) an information-rich frontend from which the user can retrieve all information related to the flows at different aggregations in order to make decisions, and (*iii*) a verification process accelerated by flexible heuristic rules while leveraging human expertise and knowledge of the particular site. A detailed description for each layer is presented below.

3.1 Data Infrastructure

The data infrastructure processes traces to collect information on different levels of aggregation (packets, flows, {IP, port, proto.} tuples, and hosts). The original data source is a full payload packet trace which contains the finest grained form of information, i.e., the payloaded packets.

The trace is organized into files of relatively small size (e.g., 512 MB) and indexed by the timestamp of the first packet contained in each file. In addition, a secondary index is created for each file by indexing each packet using its IP protocol, and source and destination addresses. This two-level hierarchical index has proved to be very efficient for supporting time interval queries within large collections of data and arbitrary flow lookup during the verification process (e.g., when inspection of a single flow is needed).

Firstly, the trace is processed in order to collect the information about every flow therein. The process consists of three tasks:

1. aggregating packets into flows,

2. recording a number of statistics for each flow (e.g., number of packets and bytes, packet sizes and inter-packet arrival times), and

3. pattern matching with known application signatures of the payload content of the first few packets of a flow.

This information is exported into a database table (*Flows* table) where an entry is created for each flow. Then two further tables are created based on the *Flows* table: namely the *Hosts* table and *HostPorts* table, to support aggregated views of the traffic grouped by server or client IPs. These views enable the user to browse the general behavior on a higher aggregation and also to facilitate the verification of traffic at this level.

The tables are indexed by many of their fields such as IP addresses, port numbers, signature-matching results, etc., in order to drastically reduce the time to lookup information from the database. This enhances the interactivity of the frontend.

As for the implementation, the data infrastructure makes use of standard data formats (e.g., pcap traces) and databases systems (e.g., MySQL, BerkeleyDB). In addition, it uses Click [12, 13] as the default packet processing tool plus a number of custom programs and scripts that tie the entire structure together.

3.2 The Verification Frontend

This second layer of GTVS consists of a web-based frontend that offers mechanisms to facilitate the verification process. This includes a graphical interface which presents abundant information at various levels of aggregation, and supports the use of different kinds of heuristic rules.

The information presented in the verification frontend is collected from a broad set of sources. This collection virtually combines the merits of many previous traffic classification works, including packet payload information (as in [5]), flow statistics (as in many behavioral traffic classification methods [6, 14, 15]), host-level connection statistics (as in [7]), host name information [16], and transport-layer behavior of the P2P overlay networks [17].

By default, the verification frontend provides:

- Basic information about a flow (e.g., duration, number of packets, total number of bytes and payload bytes in each direction and TCP flags).
- Breakdown of signature matching results from a fine-tuned set of signatures.
- Host name resolution for all the IP addresses appearing in the trace[2].
- Port-based mapping using a comprehensive list of well-known port numbers.
- Packets' payload content (e.g., tcpdump of a flow).
- Visualization of the peer connection graph of an overlay network.

[2]Ideally, the IP addresses should be resolved at the time when the trace is collected. However, for previously collected traces, host names can be mined from the DNS traffic in the original trace as well as the Host header field in the HTTP requests, or, in the worst case, resolved when the trace is being verified.

```
do_heuristic1('eDonkey', "L7Marks LIKE '%eDonkey%' AND Flows >= 5",
    ['NC+10', 'NC-10', 'eDonkey'], 0.8) {|aggId|
    GTVS.aggVerify(trace, aggId, 'eDonkey', 'P2P')
}
```

Figure 2: Sample script of a user-defined heuristic rule.

Additionally, further information may be available as an extension, such as data mined from the payload of flows, flow-behavior features as used in [14], or from external modules (e.g., NIDSes, specific traffic analyzers, commercial traffic classifiers and Web searching plugins).

3.3 Heuristic Rules

The verification frontend also supports the use of heuristic rules to determine the ground truth of a number of flows at a higher aggregation. The main idea is to leverage a core set of automated procedures to verify subsets of similar flows with very high confidence, while resorting to human discernment when not enough clues are available to recognize the nature of certain flows.

The heuristics can either be derived empirically or built up using a combination of signature matching results, and *a priori* knowledge about known services, port numbers and host names. The user can flexibly build his own heuristic set, blending his own site and application-specific knowledge to facilitate desired tasks. Custom heuristics can be written in the form of simple scripts that interact with the GTVS API, as illustrated in Figure 2. Then he needs to validate the heuristics using the verification frontend, or using a specific dry-run mode which is available for previewing the results of an action before actually modifying the database. On applying heuristic rules, GTVS will search for potential candidate flows and verify those which satisfy the conditions given in the heuristics.

In our experience, the use of heuristic rules has allowed us to drastically reduce the time needed to verify the ground truth.

4 Accelerating the Ground-Truth Verification with GTVS

The use of GTVS does not replace the manual verification process but is dedicated to accelerating it.

Here we suggest two principles, namely those of efficiency and accuracy, which we apply to the use of GTVS. The efficiency principle is to always try to work upon higher aggregations (e.g., services rather than individual flows) whenever possible. For example, large numbers of well-known, traditional service traffic on a specific host can be verified in the first instance. The accuracy principle is to make decisions only with very high confidence, e.g., when strong evidences from two or more mutual-independent information sources match with each other.

Distinct IPs	Server IPs	Server IP:port pairs	Client IPs	Flows	Packets	Bytes
25,631	11,517	12,198	14,474	250,403	10.9 M	7.4 GB

Table 1: Working dimensions of our data set.

Normally, the hand verification of an hour-long trace on a 1 Gbps link would take more than a hundred man-hours [5]. With GTVS, we hope an experienced user would be able to verify an initial data trace within days. Furthermore, based on the knowledge derived from the initial data, the traffic on the same site for successive periods could be verified in a fraction of the initial time.

In this section, we use the case study of a 30 min trace as an example to introduce the heuristic rules and show how they are exploited to accelerate the verification process. The trace was collected in mid December 2007 from the link to the Internet of a research facility. There were several thousands of users on site, mainly researchers, administrators and technical staff. Table 1 lists the working dimensions of our data set.

Within this half-hour trace, we focus on describing how we verified the complete TCP flows, i.e., the flows that are captured entirely from triple handshake to tear down. As for the rest, the UDP flows are verified in a much similar way, except that they are defined using a configurable timeout value. The incomplete TCP flows are typically composed of various kinds of scans and unsuccessful connection attempts. Most of this traffic has distinguishable patterns upon which custom heuristic rules can be built up.

4.1 Initial Application Clues

A set of payload signatures is used in GTVS to help collect the clue of an application from packet payloads. Our signature set is capable of identifying 35 most popular protocols. These signatures are derived from databases available on the Web, (e.g., L7-filter [18], while a substantial number of improvements and adjustments were made to enhance the accuracy and to reduce the overmatching conditions. We tested the signatures on previously hand-classified data and several segments of new data. The underspecified signatures which create many false positives (e.g., eDonkey, Skype) have either been changed or excluded, while the undermatching ones (e.g., BitTorrent) have been improved. Of course, the signatures are still far from being able to identify the totality of the traffic. However, they can be regarded as a useful clue, especially when the results they provide can be co-validated with different evidence.

4.2 The Verification Process (in Iterations)

Our approach is based on a number of successive iterations, each refining the result. Each successive classification iteration focuses upon the remaining unclassified flows about which we have the most confidence. In this way, we can accumulate knowledge based on the highest level of confidence and use this to assist in the classification of data about which we have lower levels of confidence.

Our approach requires the grouping of the heuristics to each iteration and then ordering of the iterations based upon the level of confidence we are able to place in the classification outcome.

We have derived a set of heuristics of which a core subset is presented here. We consider this subset contains those heuristics that provide sufficient generality to be useful for a wide range of applications across different physical sites.

First iteration. Based on the assumption introduced and justified in section 2, we derive some simple heuristics below.

If the signature matching results for a specific endpoint (i.e., a TCP or UDP server:port pair) appear to be strongly consistent, we can reasonably assume that we have identified a particular service on that endpoint. As already noted, there are a few exceptions to this assumption, but, it works well for many protocols, especially for those with a well-established signature (e.g., text-based protocols). Several criteria are used to quantitatively justify the consistency: thresholds are specified to guarantee that at least a certain percentage of flows as well as a minimum number of flows have matched a specific signature. In addition, only a given subset of signatures is allowed to have matched the flows. For example, it is known that HTTP might appear in BitTorrent traffic as certain clients use it in some of their communications, but BitTorrent should not appear in the flows toward a Web server. This constraint is expressed by defining a subset of possible signatures (which does not include BitTorrent when the heuristic is used for HTTP traffic). The thresholds are initially set in a conservative way (e.g., at least 90% and 10 flows), and will be tuned in the third iteration. We apply this heuristic for most of the protocols.

The next heuristics are based on the assumption that flows between the same IP addresses pair are likely due to the same application. For example, FTP traffic between two hosts can be easily verified by considering a host that has an already verified FTP service (e.g., using the first heuristic) and a number of flows each going to a different high port number on the same destination address in an incremental fashion. As another example, consider the HTTPS protocol. In many cases a Web server is running both standard and secure HTTP services. If a standard HTTP service has been verified on port 80, and a certain fraction of flows to port 443 matches the SSL signature, then the flows between a hosts pair can be heuristically verified. Other similar heuristics can be derived for streaming and VoIP applications: for example, RTSP appear within a TCP control channel while the data are relayed on a unidirectional UDP stream; instead a VoIP application may use a SIP session and a bi-directional UDP stream.

Second iteration. A great amount of information can be derived from the host names. For very popular services (e.g., Google, MSN, eBay), heuristics based on domain names can be easily defined: e.g., HTTPS traffic to MSN servers is due to MSN messenger instead of browsers as well as traffic on port 1863. Further, assuming the trace was captured at the edge of a certain network, specific site information such as the internal services and traffic policies, can be used to efficiently verify a part of the traffic.

Third iteration. Now we try to lower the thresholds of the previous heuristics by inspecting the flows on particular hosts which do not match the signature. If those flows also correspond to the same service application, the threshold can be lowered for those hosts.

Fourth iteration. In this iteration we consider behavioral characteristics of hosts in regard to overlay networks, mainly for the identification of P2P traffic. A typical example, however, is the SMTP traffic which has a strong P2P-like behavior in that SMTP servers act as both the recipient and the sender of emails. The assumption is that if a host is an SMTP server, all the flows generated from this host toward port 25 are mail traffic. In general, this heuristic is applicable for P2P traffic as long as the information about the port number can be utilized[3] and the assumption of the heuristic can be validated. In our experience, for example, there is a large number of eDonkey flows which can be identified using port 4662 and for BitTorrent on the port 6881. It is also possible to use this heuristic for instant messaging networks, for example, Windows Live Messenger uses the port 1863. This heuristic can re-iterate through the data set until no new flows are discovered.

Additionally, for P2P applications that use dynamic port numbers, we resort to a heuristic that considers the host activities and the relationship with the a certain application's overlay network. The idea here is to select an initial set of peers which are known to run a certain P2P application. For example there are already some P2P nodes identified in the first iteration, and some Skype clients identified in the second iteration using the information of centralized login servers Then, for each identified peer, we consider the set of hosts that it communicates with. We select the subset of hosts corresponding to the intersection of all these host sets. Lastly, we identify hosts that are likely peers by applying a threshold (e.g., ≥ 3) on the host's connection degree (i.e., how many peers are connected to this host) and selecting those hosts that do not have other conflicting activities (e.g., if they run multiple P2P applications). For Skype, we also consider whether a host receives TCP connection on ports 80 and 443 plus a high number chosen at random, as this is a strong peculiarity of this application.

Fifth iteration. From our experience, at this iteration only a small number of payloaded flows remain. User-defined heuristics can be derived according to the specific applications in the analyzed trace, or to the particular behaviors that might be found through manual inspection.

Also, based on information of the already verified hosts in the data set, one can start to label the incomplete TCP flows and unknown UDP flows, using the assumption on the sub-tuple consistency in Section 2.

So far, the heuristic rules have greatly reduced the time to verify the data, although manual verification of a small amount of remaining traffic is still necessary, especially for the identification of new applications.

Table 2 summarizes the total traffic breakdown as we verified it. Table 3 shows the partial results measured at the end of each iteration, which are graphically presented in Figures 3 and 4 for two metrics: flows and bytes, respectively. Figure 5 reports the evolution of completeness for clients and servers during each iteration. As can be seen, a very small number of clients are responsible for the 2,041 unknown flows toward 1,736 servers. In this case, these hosts happen to simultaneously run several P2P applications and we are not able to determine a final conclusion on the specific application.

Finally, we evaluate L7's accuracy based on the obtained ground truth. Table 4 shows per-class false negatives and positives. Signatures do not significantly over match, yielding to very few false positives. However, with the exception of web-browsing class, all classes exhibit many false negatives. This is due to two major factors: underspecified signatures and obfuscated traffic. In both cases, our method can exploit information about traffic aggregates to derive the actual application and produce accurate ground truth.

[3]Despite many P2P hosts using non-standard port numbers, we still observe that many P2P nodes still use the well-known port numbers.

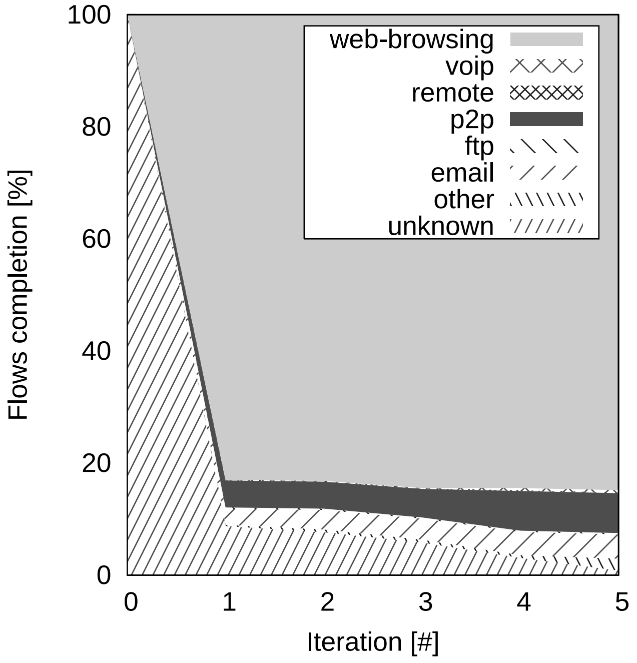

Figure 3: Verification completeness against successive iterations — breakdown of verified flows.

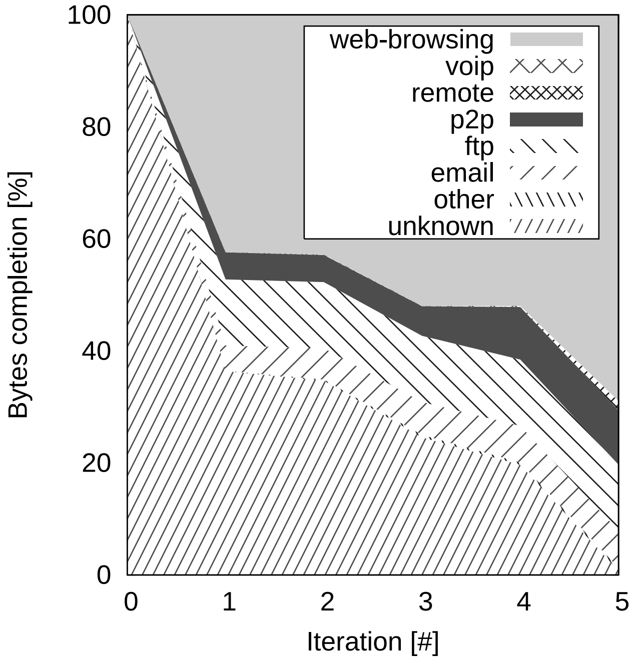

Figure 4: Verification completeness against successive iterations — breakdown of verified bytes.

Class	Flows	Packets	Bytes [MB]
email	10,871	808,272	470.55
ftp	555	894,805	838.22
gaming	150	2,882	0.47
im	506	21,036	4.18
malicious	4,008	62,259	5.30
p2p	17,851	1,125,766	685.43
remote	317	135,735	109.26
services	618	16,675	9.22
streaming	11	17,815	16.33
voip	1,043	52,020	11.92
web-browsing	212,432	7,630,649	4,889.80
unknown	2,041	112,840	52.66

Table 2: Traffic breakdown by class.

Class	Number of flows by iterations				
	1st	2nd	3rd	4th	5th
email	7,225	8,743	9,439	10,420	10,871
ftp	555	555	555	555	555
gaming	0	108	108	108	150
im	65	503	505	505	506
malicious	0	0	0	0	4,008
p2p	12,046	12,046	12,728	17,708	17,851
remote	254	254	254	254	317
services	466	604	610	610	618
streaming	0	2	8	8	11
voip	0	121	121	1,042	1,043
web-browsing	207,888	208,313	211,522	211,522	212,432
unknown	21,904	19,154	14,553	7,671	2,041

Table 3: Evolution of verification completeness by iteration.

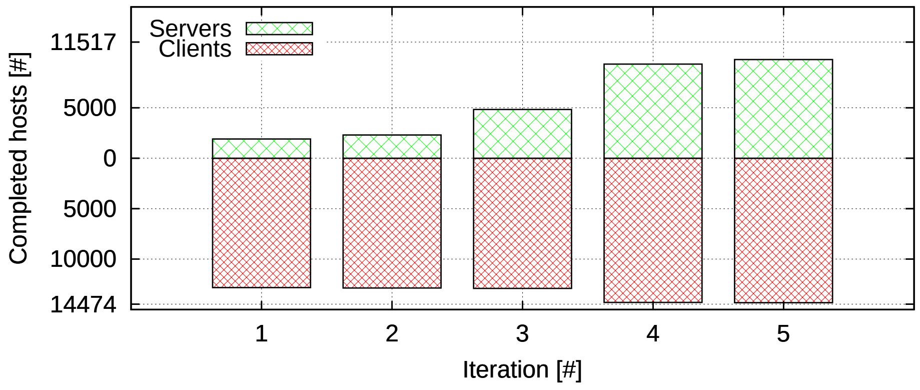

Figure 5: Verification completeness of clients and servers against successive iterations.

Class	False negatives [%]			False positives [%]		
	Flows	Packets	Bytes	Flows	Packets	Bytes
email	25.99	22.33	21.75	0.00	0.00	0.00
ftp	81.26	99.53	99.97	0.00	0.00	0.00
gaming	100.00	100.00	100.00	0.00	0.00	0.00
im	74.90	79.60	81.78	0.00	0.00	0.00
malicious	100.00	100.00	100.00	0.00	0.00	0.00
p2p	16.75	16.65	14.67	0.34	1.85	2.10
remote	18.93	0.52	0.04	0.00	0.00	0.00
services	98.54	99.48	99.94	0.00	0.00	0.00
streaming	27.27	0.15	0.02	0.00	0.00	0.00
voip	100.00	100.00	100.00	0.00	0.00	0.00
web-browsing	0.29	0.42	0.40	0.42	0.52	0.27

Table 4: Evaluation of L7's per-class accuracy.

4.3 Discussion

Here, we have focused on describing the verification of application traffic. The verification processes of malicious and unwanted traffic can similarly be based on their specific patterns.

One can see that the first-time use of GTVS on any given trace will often require the inspection of small segments of data throughout the process, in customizing and testing new heuristics, dry-runs, tuning thresholds and final manual decisions on hard objects. However, if one is carrying out a continuous ground truth collection work on a specific site or on many sites simultaneously, time would be further saved as we expect only limited tuning and validation are needed.

Since this framework will become publicly available, we envision it will also be easier to share the knowledge within the community: not only the protocol signatures but also the heuristics and application-specific knowledge will become a public resource.

We also note that the confidence of the ground truth verified by GTVS relies mainly on its user. Therefore to collect good ground truth requires sufficient user interactions and dry-runs to double-confirm the user's judgments.

5 Classifying HTTP Traffic

The Hyper-Text Transfer Protocol (HTTP) is perhaps the most significant protocol used on the Internet today. Traditionally, the HTTP has played a fundamental part in the Web-browsing activities (e.g., publishing, searching, advertising), however the growth of network computing, web applications and services have expanded its role beyond user-driven web browsers, while increasing the number of applications that use HTTP for a variety of functions.

This is largely thanks to the request-response paradigm between a client and a server for which HTTP was defined. Such a paradigm is robust and flexible enough to allow most kinds of data transmission tasks besides the fundamental Web-browsing activities. Also, a HTTP client is handy to implement and can be easily embedded in any software

Data Set	Web servers	Distinct client applications	HTTP connections	HTTP bytes [MB]
Day1	2137	113	50,002	695
Day3	2509	232	105,813	1,950

Table 5: Number of servers, applications, connections and bytes in the traces.

to display advertisements or to send or retrieve data in a light-weighted way. Such dual property of HTTP has led to a rise of several phenomena, such as:

Non-Web Activities over HTTP. HTTP has accommodated remarkably more different kinds of activities than Web-browsing, for example sending and receiving email, file downloading and sharing, instant messengers and multimedia streaming, as long as they can follow the request-response paradigm. Some of them were using a web interface, while many others blend the traffic into HTTP traffic in order to gain market advantages, data transmission priority or at least to be allowed by firewall rules.

Embedded HTTP Clients. Applications not related to web-based activities would also use HTTP for a secondary function, for example: advertiser sponsored software retrieving advertisements or an operating system automatically checking for updates.

Unlike works that characterize an individual new component in the HTTP traffic such as [19], we are interested to explore the increasing variety of application activities over HTTP, such as advertising, crawlers, file downloads, web-based applications and news feeds, and analyze the evolution of the usage of HTTP. From these analyzes, we aim to (*i*) understand the growth of the HTTP traffic, (*ii*) illustrate in depth how HTTP is being used on the current Internet and (*iii*) describe the use of payload indicators that can reveal the actual purpose of an HTTP connection. We consider this work as a contribution to the understanding of the Internet evolution through a measurement study. In common with [19], we conclude that this work has to be done on a continual basis. A clear opportunity exists to spot the future trend in the traffic models, social network models and economic models of the next generation Internet.

5.1 Method

Data

To analyze the application activities within the HTTP "traffic-mix" as a whole, we collected a half-hour trace at around 11 AM of a working day in late 2006 (referred to as Day3) from the edge of a research-institute network and compare it with a trace collected at a similar time from the same network in 2003 (called Day1). The research facility has about 1,000 users and is connected to the Internet via a full-duplex Gigabit Ethernet link. We monitored the bi-directional traffic on its link to the Internet. We study the connections over HTTP, excluding connections which use an extended HTTP protocol specific for some P2P file sharing applications. The HTTP traffic is identified based on packet content and host knowledge using GTVS, regardless of port numbers. The traces are listed in Table 5. The number of distinct client applications is estimated from the user-agent field and excluding different versions of the same application.

Class	Activities	Signatures from
Web-browsing	visiting web pages using a web browser	Method + URL + User Agent + Content Type
Web app	applications via web interface: e.g., Java applets, web gadgets, and a variety of software	URL + User Agent
Crawler	bots crawling web pages	User Agent
File download	file downloading over HTTP	URL + Content Type + User Agent
Webmail	web-based e-mail service	Host name
Advertising	advertisements on a web page or embedded in software	Host name + Content Type
Multimedia	streaming media or viewing media files on web pages	URL + User Agent
SW update	software update over HTTP (both automated and manual)	Content Type + User Agent
News feeds	RSS feeds	URL + Content Type
Link validator	automated link validators	User Agent
Calendar	calendar application based on web, e.g., ical, gcal	User Agent, Host name
Attack	malicious traffic over HTTP	User Agent + Url
IM	Windows Live Messenger	User Agent
Monitoring	network monitoring	User Agent

Table 6: Heuristics to identify different HTTP traffic classes.

Classification

Within the traces, we discovered many different applications and purposes over HTTP. We try to categorize the traffic into a number of classes based on its purpose. There are several key fields in HTTP headers that can reveal the client and the activity information of a connection, including the request method, host name, URL of the target object (which often includes the file name of the object), content type, and the user-agent [20]. An individual field may not be sufficient: for example, each different type of user-agents tells us of a different application, but there are many applications which tend to use default browser settings. However, through manual payload inspection, we are able to derive a number of heuristics to classify the majority of the traffic using a combination of the signatures in these fields. These signatures are manually derived from trace payload and online resources such as [21] and will have minimum false positives but may contain some false negatives for other categories than Web-browsing. Table 6 describes the activity classes as well as the signatures we use to identify them.

5.2 Results

We compared the total bytes in each HTTP traffic classes in the two traces, as shown in Table 7. While the whole HTTP traffic increased by 180%, Web-browsing and Crawler both increased by about 108%. However, several classes have seen a sharp rise: Web app, File download, Advertising, Webmail, Multimedia, News feeds and IM. Further, the Day3

Class	Day1		Day3		Day1:Day3
	[KB]	[%]	[KB]	[%]	
Web-browsing	506,109	71.11	1,052,879	52.71	1:2.08
Web app	22,063	3.10	317,204	15.88	1:14.38
Crawler	143,440	20.15	296,048	14.82	1:2.06
File download	24,817	3.49	232,878	11.66	1:9.38
Webmail	5,418	0.76	43,319	2.17	1:8.00
Advertising	3,804	0.53	28,962	1.45	1:7.61
Multimedia	1,949	0.27	10,627	0.53	1:5.45
SW update	3,583	0.50	8,140	0.41	1:2.27
News feeds	336	0.05	6,108	0.31	1:18.18
Link validator	157	0.04	439	0.02	1:2.79
Calendar	0	0.00	324	0.02	1:∞
Attack	0	0.00	232	0.01	1:∞
IM	1	0.0001	211	0.01	1:222.82
Monitoring	24	0.003	26	0.001	1:0.92

Table 7: Traffic volume comparison between Day1 and Day3.

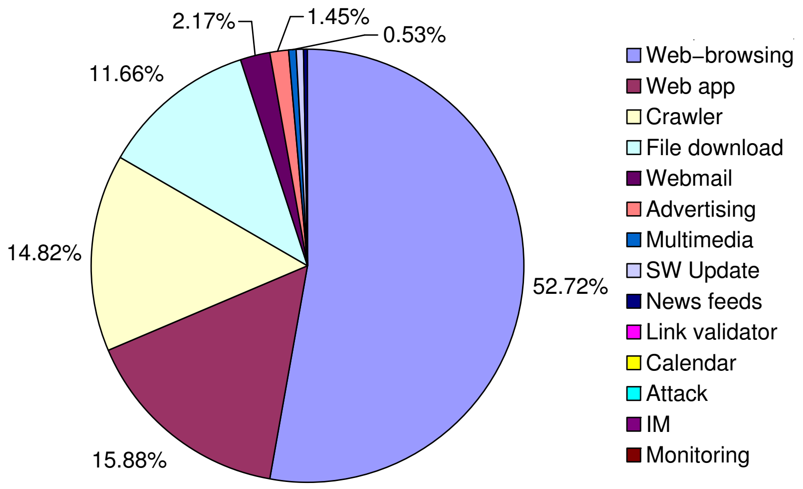

Figure 6: Day3 HTTP traffic breakdown.

HTTP traffic breakdown is shown in Figure 6, in which the Web-browsing component only constitutes of no more than 53% of the HTTP traffic.

6 Related Work

The lack of carefully verified traffic traces is publicly recognized as a major obstacle for much interesting research. A recent call to arms has been issued by Salgarelli *et al.* in [22].

On the technical aspects, our work can be seen as a cumulative progress, with lots of inspirations from previous traffic classification works, including [6, 7, 14, 16, 17, 23]. Each of these works made use of a different set of information sources, which are combined in our framework.

A content-based classification scheme comprising of nine identification methods was presented in [5]. Despite their highly accurate and complete results, there was not a systematic infrastructure or an indication of how the procedure can be efficiently organized. Thus a barrier still exists preventing other people from repeating their method. Further, the work presented here uses a broader set of information sources.

In [24], the authors suggested a technique based on active measurements to cover the shortage of ground-truth data. Their system includes a modified network driver (to be executed on the end-hosts) that extends outbound IP packets with the Router Alert option field. This field, which is said to be transparent for routers and end-hosts, is used to mark each packet with the first two characters of the corresponding program name. The mark makes it possible to differentiate traffic using just a packet header field. However, two characters of the program name are not enough to avoid name aliasing that would be very undesirable for application-specific service differentiation. Furthermore, the marking operation increments of four bytes all packets except those whose size is the MTU (Maximum Transfer Unit). Therefore, the resulting traffic is altered and classifiers trained from this traffic will implicitly see untrustworthy packet sizes for one direction of the flows.

Specific to the classification of malicious traffic through pattern recognition techniques, is the work presented in [25]. The authors described a self-training architecture for automatically building a database of labeled traffic traces. They also showed that NIDSes trained on such a database perform approximatively as well as the same systems trained on correctly labeled data. However, in their method each packet is labeled either as normal or as an attack. This is not sufficient to express the multi-class traffic composition for application identification purposes, although their approach might be used as an additional information source in our methodology.

7 Conclusions

In this paper, we presented the novel Ground Truth Verification System (GTVS). A detailed guide is shown on how to use GTVS to accelerate the verification process, as well as the results by iterations from a case study of real traffic. Further, we are publicily releasing this system and our rule sets at `http://www.cl.cam.ac.uk/research/srg/netos/brasil/`.

It is hoped that it will substantially save the time and labor for individual researchers, and more public data with ground truth labels may subsequently become available to the community in the near future.

References

[1] Kim, H., kc claffy, Fomenkova, M., Barman, D., Faloutsos, M.: Internet traffic classification demystified: The myths, caveats and best practices. In: Proceedings of the Fourth ACM Conference on Emerging Network Experiments and Technologies (CoNEXT'08). (Dec 2008)

[2] LBNL: The internet traffic archive `http://ita.ee.lbl.gov/html/traces.html`.

[3] CAIDA: Data collection at caida `http://www.caida.org/data/`.

[4] MAWI: Working group traffic archive `http://mawi.wide.ad.jp/mawi/`.

[5] Moore, A.W., Papagiannaki, D.: Toward the accurate identification of network applications. In: Proceedings of the Sixth Passive and Active Measurement Workshop (PAM'05). (Mar 2005) 41–54

[6] Moore, A.W., Zuev, D.: Internet traffic classification using bayesian analysis techniques. In: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. (2005) 50–60

[7] Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: multilevel traffic classification in the dark. In: Proceedings of the 2005 ACM SIGCOMM. (2005) 229–240

[8] Erman, J., Mahanti, A., Arlitt, M.: Traffic classification using clustering algorithms. In: Proceedings of the Second Annual ACM Workshop on Mining Network Data (MineNet'06). (Sep 2006) 281–286

[9] Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proceedings of the Thirteenth USENIX Conference on System Administration (LISA'99). (1999) 229–238

[10] Paxson, V.: Bro: a system for detecting network intruders in real-time. Computer Networks **31**(23–24) (1999) 2435–2463

[11] Dusi, M., Crotti, M., Gringoli, F., Salgarelli, L.: Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. Computer Networks **53**(1) (Jan 2009) 81–97

[12] Click: The click modular router `http://www.read.cs.ucla.edu/click/`.

[13] Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F.: The Click modular router. ACM Transactions on Computer Systems (TOCS) **18**(3) (2000) 263–297

[14] Li, W., Moore, A.W.: A machine learning approach for efficient traffic classification. In: Proceedings of 15th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'07). (Oct 2007) 310–317

[15] Bernaille, L., Teixeira, R., Salamatian, K.: Early application identification. In: Proceedings of the Second ACM Conference on Emerging Network Experiments and Technologies (CoNEXT'06). (Dec 2006) 1–12

[16] Trestian, I., Ranjan, S., Kuzmanovi, A., Nucci, A.: Unconstrained endpoint profiling (googling the internet). In: Proceedings of 2008 ACM SIGCOMM. (Aug 2008) 279–290

[17] Karagiannis, T., Broido, A., Faloutsos, M., kc claffy: Transport layer identification of P2P traffic. In: Proceedings of the Fourth ACM SIGCOMM Conference on Internet Measurement (IMC'04). (Oct 2004) 121–134

[18] L7-filter: Application layer packet classifier for linux `http://l7-filter.sourceforge.net`.

[19] Schneider, F., Agarwal, S., Alpcan, T., Feldmann, A.: The new web: Characterizing AJAX traffic. In: Proceedings of the Nineth Passive and Active Measurement Conference (PAM'08). (Apr 2008) 31–40

[20] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: RFC 2616: Hypertext transfer protocol – HTTP/1.1 (Jun 1999)

[21] of User-Agents, L.: `http://www.user-agents.org/`.

[22] Salgarelli, L., Gringoli, F., Karagiannis, T.: Comparing traffic classifiers. ACM SIGCOMM Computer Communication Review **37**(3) (Jul 2007) 65–68

[23] Dreger, H., Feldmann, A., Mai, M., Paxson, V., Sommer, R.: Dynamic application-layer protocol analysis for network intrusion detection. In: Proceedings of the 15th conference on USENIX Security Symposium (USENIX-SS'06). (Aug 2006) 257272

[24] Szabó, G., Orincsay, D., Malomsoky, S., Szabó, I.: On the validation of traffic classification algorithms. In: Proceedings of the Nineth Passive and Active Network Measurement Conference (PAM'08). (Apr 2008) 72–81

[25] Gargiulo, F., Mazzariello, C., Sansone, C.: A Self-Training Approach for Automatically Labeling IP Traffic Traces. In: Computer Recognition Systems 2. Volume 45 of Advances in Soft Computing. Springer-Verlag (2008) 705–717