POLITECNICO DI TORINO
Repository ISTITUZIONALE

WHAT: A Big Data Approach for Accounting of Modern Web Services

Original
WHAT: A Big Data Approach for Accounting of Modern Web Services / Trevisan, Martino; Drago, Idilio; Mellia, Marco; H. H., Song; Baldi, Mario. - ELETTRONICO. - (2016), pp. 2740-2745. (Intervento presentato al convegno IEEE Workshop on Big Data and Machine Learning in Telecom (BMLIT) tenutosi a Washington nel December 2016) [10.1109/BigData.2016.7840921].

Availability:
This version is available at: 11583/2656557 since: 2017-06-08T14:21:39Z

Publisher:
IEEE

Published
DOI:10.1109/BigData.2016.7840921

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

29 March 2024

WHAT: A Big Data Approach for Accounting of Modern Web Services

Martino Trevisan† Idilio Drago† Marco Mellia† Han Hee Song‡ Mario Baldi†‡
†*Politecnico di Torino, Italy* ‡*Cisco, Inc.*

***Abstract*—HTTP(S) has become the main means to access the Internet. The web is a tangle, with (*i*) multiple services and applications co-located on the same infrastructure and (*ii*) several websites, services and applications embedding objects from CDN, ads and tracking platforms. Traditional solutions for traffic classification and metering fall short in providing visibility in users' activities. Service providers and corporate network administrators are left with huge amounts of measurements, which cannot immediately reveal the real impact of each web service on the network. Such visibility is key to dimension the network, charge users and policy traffic. This paper introduces the Web Helper Accounting Tool (*WHAT*), a system to uncover the overall traffic produced by specific web services. *WHAT* combines big data and machine learning approaches to process large volumes of network flow measurements and learn how to group traffic due to pre-defined services of interest. Our evaluation demonstrates *WHAT* effectiveness in enabling accurate accounting of the traffic associated to each service. *WHAT* illustrates the power of machine learning when applied to large datasets of network measurements, and allows network administrators to regain the lost visibility on network usage.**

I. Introduction

Monitoring how web services are used and how they consume network resources is key to Internet Service Providers (ISP) and network administrators when operating and planning the network. Companies, for instance, need to monitor their enterprise traffic to limit bandwidth consumption, spot sudden growth in usage of services, and enforce corporate polices on accredited services. With more and more enterprise traffic directed to web applications offering IT services, network managers have now an urgent need for tools to understand and control network usage.

Traffic classification plays a fundamental role in uncovering what applications and services are being accessed, and a variety of classification methods has been proposed in the past [1], [2]. A large and growing fraction of transactions happening over the Internet is based on the HTTP(S) protocol nowadays. Whether users are browsing the web, accessing business or leisure applications, using mobile apps, sharing or accessing content, chances are HTTP(S) is used to support the communication. The clear trend towards encryption by default [3] leaves in-network monitors with large collections of raw data, mostly containing layer-3 and layer-4 flow information, which are insufficient to accurately reveal how applications consume network resources.

This research has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, "BigDAMA".

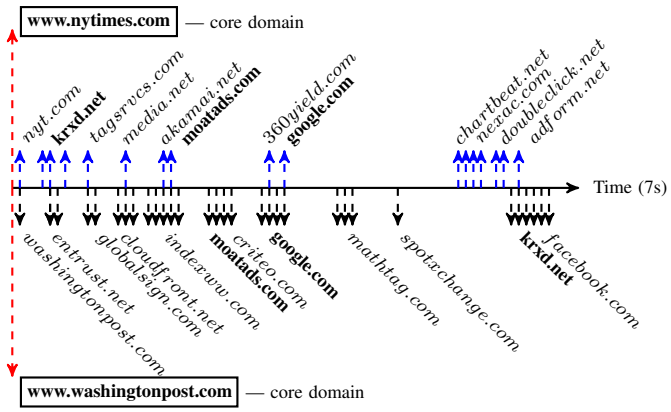


Figure 1: Flows opened when visiting $nytimes.com$ and $washingtonpost.com$ – in bold, shared third-party services.

Augmenting flow information with the name of servers, as obtained via DNS [4], [5] or TLS handshake parsing, has been proposed as a means to overcome such limitations. By relying on server names indications, in-network monitors could infer the service or organization behind each flow. Unfortunately, the widespread use of Content Delivery Networks (CDNs), as well as ads and tracking platforms, challenges the methodology – e.g., because (*i*) CDNs and cloud platforms co-locate multiple services or (*ii*) different websites, services and mobile applications generate HTTP(S) flows to similar servers to retrieve third-party content, ads, trackers etc.

The question we want to address is *what is the total cost of visiting a given site, including all traffic the client downloads due the visit*. An example of the difficulties to address this question is shown in Fig. 1. Assume a user visits two news websites. Flows observed in the network are depicted as arrows, which are annotated with the 2nd-level domain name of servers. Most names are not informative, and both sites contact common servers (in bold) to render pages. We are interested in accounting all flows as *triggered by* the original sites (i.e., the red arrow). We call these sites the *core domains* the user intentionally visit. A naive methodology taking into account only the flows to core domains would identify less than 20% (4%) of the actual bytes (flows) caused by the visits, whereas numbers would be increased to 70% (30%) if domain ownership – e.g., *nyt.com* belongs to *nytimes.com* – is considered.

We apply machine learning to address the challenges of precisely accounting web traffic. We present the *Web Helper Accounting Tool (*WHAT*)*, a system to automatically learn which flows are triggered by visits to a website. *WHAT* is completely unsupervised. It learns dependencies from flow-level traces annotated with the domain names. Given a list of core domains of interest, *WHAT* automatically identifies the *support domains* that are opened for downloading pictures, plugins, etc. Since support domains may serve many websites, *WHAT* includes mechanisms to identify the most likely core domain triggering each observed flow.

WHAT identifies subordinate flows by creating *Bag of Domains (BoDs)*: a model of the traffic generated by accessing a site, based on the unordered set of all support domains that may be triggered by the core domain visit. Ingenuity is required to weight support domains and avoid background traffic to pollute BoDs. *WHAT* successfully adopts text processing approaches to obtain representative BoDs.

Our contribution is a fully working system capable of applying the technique on flow-level traces. Given the typical large volumes of such traces, *WHAT* must rely on state-of-the-art "big data" platforms and is implemented using the Apache Spark framework. *WHAT* is validated using actual browsing histories of 30 volunteers and then applied to a 2 month long dataset collected from a live ISP network to provide an indication of the system performance.

II. The WHAT System

A. Architecture Overview

We assume a passive network monitoring infrastructure is in place and exposes per-flow records to be classified by *WHAT* (e.g., NetFlow, or logs collected by proxies) according to the website that triggers them (see Fig. 1). Beside traditional information such as flow identifiers, client identifiers, traffic volume and timestamps, we assume each flow is already annotated with the Fully Qualified Domain Name (FQDN) of the server being contacted, hereafter informally called domain name [4], [5].

WHAT is a completely unsupervised system. It builds a model based on flow traces and then uses it to classify traffic. *WHAT* defines the model in a completely automatic way, minimizing user intervention and adapting to usage scenario.

Fig. 2 summarizes the *WHAT* architecture. It is composed of two modules: The *Training Module* and the *Classifier*. A list of user-provided core domains are provided as input. *WHAT* learns the BoD for each core domain from traffic traces. BoDs are then employed to classify new flows from live traffic. We next describe the expected input data format, followed by the working internals of *WHAT* modules.

B. Input Data

WHAT expects two input files. First, the user must provide a list of targeted websites – i.e., a list of *core domains*. Such list is a set of domain names that can be retrieved from public repositories (e.g., open crawling efforts [6]), or can be manually crafted to include only services of interest.

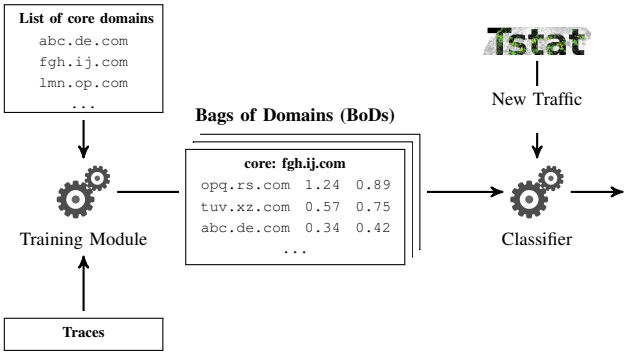


Figure 2: *WHAT* architecture.

Secondly, *WHAT* must receive flow records, such as, for example, those exported by NetFlow. Given a flow f – i.e., an entry composed of client and server IP addresses, port numbers and transport protocol – let ts_f, te_f be the time of the first and last packet in the flow. We assume that the flow record is enriched with information about the server domain name d_f used by clients when obtaining the server IP address. Flow meters typically export information from the network and transport layers, missing the association between IP addresses and domains. Different methods can be used to annotate flow records with domains. DNS logs can be employed to extract queries/responses and annotate records either online [4], [5] or in a post-processing phase. Equally, some flow meters export domain names on-the-fly – e.g., extracting Server Name Identification (SNI) from TLS flows, or the `Host:` field from plain HTTP headers.

We rely on Tstat [7] to collect data summarizing flows. Tstat exposes more than 100 metrics, including the typical ones exported by popular flow meters – i.e., server IP addresses contacted by clients, flow timestamps and bytes counters. Also, Tstat implements all the above mechanisms to extract server domain names and label traffic flows.

C. Training Module

WHAT training consists of building a BoD B_c for each core domain c. This step is challenging because domain names observed in the network may fall into many categories, such as: (i) core domains passed as input; (ii) support domains that are triggered by multiple core domains; (iii) unknown domains – e.g., domains contacted by background services; (iv) *false* core domains – e.g., domains that are both core and support domains, depending on the website being visited. For instance, we can see in Fig. 1 that flows to *google.com* are opened by both news websites. In this example, *google.com* is a support domain for *nytimes.com* and *washingtonpost.com*. The same domain name is however a core domain used to host Google services.

WHAT relies on two ideas to build BoDs from traces. First, it learns BoDs by observing flows that start near in time to core domain flows. The intuition is that support flows are opened immediately after a visit to websites. We call the time interval in which flows are considered the *observation window* OW. Second, *WHAT* filters out unrelated domains by calculating a *domain score*, represented by frequency that support flows are seen in observation windows.

1) **Observation Window and BoDs:** Given the set of core domains C, *WHAT* learns the BoD B_c for each $c \in C$ *WHAT* considers the flow traces generated by each client, e.g., all flows generated by the same client IP address. *WHAT* extracts the BoDs from passive traces directly at the vantage point, i.e., learning (and updating) the BoDs from the data the system is exposed to. While learning BoDs, *WHAT* minimizes the impact of false core domains by considering valid triggers those flows directed to a core domain c that appears after a idle period ΔT_{idle}, i.e., likely due to a new user visit.

When a trigger is observed, *WHAT* extracts all domains found in the observation window OW following it. The duration of the observation window, ΔT_{OW}, is a parameters of the system, discussed in Sec. IV. A domain d appearing in the OW becomes part of the BoD B_c as a support domain. Traces from all clients contribute to learn B_c.

Notice that not all support domains appear after every visit to a website. More dangerous, background traffic and support domains triggered by other core domains may appear in OW_c by chance, poisoning B_c with *false support domains*. *WHAT* needs then to observe a large number of OWs to accumulate support domains, and select those that are actual support domains. The assumption is that support domains emerge, whereas the irrelevant ones (e.g., background domains and false support domains) can be filtered out by means of thresholds and domains scores.

2) **Domains Score:** The key idea is that domains that are triggered by a core domain should appear more frequently over multiple OWs than other domains. We leverage text processing methodologies to implement a filtering process based on this idea. We rely on the $tf-idf$ (term frequency – inverse document frequency [8]) of domains in bags to represent the scores. The $tf-idf$ is used in information retrieval to evaluate the importance of a word to a document in a collection. A word is more important when it appears often in a document (the tf), but its importance is reduced by a factor representing how frequent the word appears in other documents in the collection (the idf).

In our problem, a document is a BoD B_c for the core domain c, a word is a domain name $d \in D$ and the collection of documents is the set of all bag of domains $BoDs$.

The training phase results in a BoD for each core domain $c \in C$. Each domain $d \in B_c$ is associated two scores:

$$B_c = \{(tf(d, B_c), tf_idf(d, B_c)) | d \in D\}. \quad (1)$$

Algorithm 1 $classify(C, BoDs, F)$

Input:
$C = \{c_1, ..., c_k\}$ ▷ *core domains*
$BoDs = \{B_{c_1}, ..., B_{c_k}\}$ ▷ *BoDs of core domains in C*
$F = \{f_1, ..., f_n\}$ ▷ *list of flows of a client to be classified*
Output:
$O = \{(f_1, l_1), ..., (f_n, l_n)\}$ ▷ *labeled flows*

1: $W \leftarrow \emptyset$ ▷ *set of currently active EVs*
2: $O \leftarrow \emptyset$
3: **for** $f \in F$ **do**
4: *// retrieve start/end times and domain name of f*
5: $ts_f, te_f, d_f \leftarrow parse(f)$ ▷ *ts_f is also current time*
6: *// remove expired EVs*
7: $W \leftarrow \{(ts, te, c_i, B_{c_i}) \in W | ts_f - te \leq \Delta T_{EV}\}$
8: *// obtain the best neighbor BoD among the active ones*
9: $w_{best} \leftarrow \{(ts, te, d, B)\} \leftarrow BestBoD(ts_f, d_f, W)$
10: **if** $d_f \in C \wedge valid_core(d_f, ts_f, w_{best}, F)$ **then**
11: *// start an evaluation window for core domain d_f*
12: $W \leftarrow W + \{(ts_f, te_f, d_f, B_c)\}$
13: $O \leftarrow O + \{(f, d_f)\}$
14: **else**
15: **if** $w_{best} \neq \emptyset$ **then**
16: $O \leftarrow O + \{(f, d)\}$
17: $te_{w_{best}} \leftarrow \max(te_f, te_{w_{best}})$ ▷ *fix boundaries*
18: **else**
19: $O \leftarrow O + \{(f, \text{“}unknown\text{''})\}$

If d appears in all BoDs, then $idf(d, BoDs) = 0$ and $tf_idf(d, B_c) = 0$, suggesting its presence is insignificant to characterize the document. Similarly, if d does not appear in any observation window in OW_c, $tf(d, B_c) = 0$ and $tf_idf(d, B_c) = 0$. *WHAT* uses the $tf(d, B_c)$ score to remove from B_c core domains that appear too infrequently, i.e., $tf(d, B_c) < MinFreq$, since those are likely to be background or false support domains. Trade-offs are explored in Sec. IV. The score tf_idf_{d,B_c} allows *WHAT* to assign ambiguous domains that appear into several BoDs during classification. In the following section we give details.

D. Traffic Classifier

Armed with core domains and their respective BoDs, *WHAT* processes traffic to classify flows. *WHAT* uses Algorithm 1 to classify each flow f. It receives the set of core domains C, the BoDs and the set of flows F generated by a client. It outputs flows annotated with core domains, or *unknown* in case no association is found.

The algorithm is based on the concept of *Evaluation Window* (EV), i.e., the time during which a support flow can appear after the observation of a core domain. The algorithm maintains a list of active EVs, W. The list grows as new core domains are observed (lines 10–13), and entries are aged out based on a timeout ΔT_{EV}, i.e., window ending time $te = \max_{f \in W} te_f$ is elapsed by at least ΔT_{EV} (line 7).

Differently from the training phase, the evaluation window duration is extended during classification. This happens when new support domains are found (line 17). The rationale is that flows to support domains may be observed long time after the core domain, since the terminal keeps downloading

objects due to a user action, e.g., scrolling a web page that triggers the download of new elements.

In case multiple active windows are alive, *WHAT* checks which is the most suitable one using the function $BestBoD()$ (returning w_{best} – line 9). Details are omitted for the sake of brevity. We checked different options, and opted for a "closest in time" criteria: *WHAT* looks for the closest active window among W, for which the domain d_f of f has a frequency above a $MinFreq$ threshold.

At last, *WHAT* has to resolve the ambiguity for names that are both support and core domains. When the domain d_f of the flow f is a core domain, and at the same time it belongs to at least one active BoD in W, *WHAT* disambiguates the situation relying on the function $vaild_core()$ (line 10). In sum, *WHAT* creates an evaluation window EV starting from the given ambiguous flow, and looking forward for flows in the EV *after* the current flow time ts_f. Then, *WHAT* calculates the sum of tf_idf scores for domains in both (i) this *future* evaluation window (i.e., considering d_f a core domain) and (ii) the best BoD in active evaluation windows (i.e., w_{best}, considering d_f a support domain). *WHAT* selects the situation producing the highest sum of tf_idf scores.

Being designed primarily for accounting, *WHAT* can tolerate small delays. *WHAT* processes groups of flows, which are ingested into the system in batches. It is however important to notice that the traffic classifier module operates on a per-flow and per-client basis and, thus, its algorithms can easily scale to large data streams.

III. Datasets

For training and testing we build upon two datasets. We learn BoDs using a passive trace collected from a large ISP network. Then, we assess *WHAT* classification performance using a dataset made by revisiting pages found on browsing histories of users, which list actually visited core domains. Thus, where we have the full ground truth knowledge.

A. ISP Trace

Our first dataset includes flow summaries exported by Tstat in a real deployment. We have instrumented a Point of Presence (PoP) of a European ISP, where ≈10,000 ADSL customers are aggregated. No ground truth is available in this trace. The ISP provides each ADSL customer (i.e., installation) a fixed IP address. Thus, by inspecting the (anonymized) client IP addresses in our dataset, *WHAT* isolates flows per ADSL installation, and use them as the per-client trace F. The trace includes information about traffic of all users' devices connected at home. We consider data of the entire months of March and April 2016, obtaining 2.2 billion flows related to around 5 million domains. Data is stored in a Hadoop cluster for scalable processing.

B. Validation Traces

To assess *WHAT* performance, we create a labeled dataset using data from volunteers. We collect browsing histories of 30 users, extracting all visited URLs directly from `SQLite` databases used by Safari, Chrome and Firefox. These are core domains, since users explicitly visited these URLs.

To obtain a set of support domains, we revisit each URL by instrumenting a Firefox browser with Selenium [9]. We let Selenium visit each URL and wait until the page is fully loaded (i.e., the `On Load` event is fired). The next URL in the list is then loaded after the browser is inactive for 1 second. Note that this could create artifacts, e.g., eventual video playback is stopped after 1 s from the experiment start.

In parallel, Tstat records flows seen in the network, saving the same information that would be available in real *WHAT* deployments. We post-process the trace to label each flow with the *core domain* that triggered it; To do this, we take into account the time when the browser requested a new page, and label the consecutive flows as *triggered by* the original visit. In total, 100,000 URLs are visited, referring to 3,759 core domains, and 9,764 support (possibly ambiguous) domains. Crawling was done in April 2016 and lasted 5 days. From these raw traces, we build two benchmarks:

1) **Web Browsing Benchmark:** It represents users continuously browsing the web. Visits are sequentially organized and assigned to the same client IP address. The original sequence in which pages are requested by each volunteer is maintained, thus mimicking users' behaviors. We produce the benchmark by post-processing traces to replace the timestamps recorded by Tstat while revisiting URLs using Selenium. Inter-visit times follow the distribution observed in browsing histories. After the time of core URL visits are determined, we populate the benchmark with support flows, respecting their inter-arrival time seen by Tstat.

2) **Concurrent Navigation Benchmark:** This benchmark simulates several browsing threads in parallel. Core and support domains of many visits appear simultaneously in the traces. This benchmark is created by repeating the previous steps, so that each thread simulates an independent (active) user. To avoid any kind of synchronization among threads, each navigation starts following the concatenated browsing histories of volunteers at a random position. This scenario can be seen as an extreme case of NAT, where n user are concurrently and continuously browsing the web.

IV. WHAT Validation

A. Classification Performance

We evaluate *WHAT* performance when classifying new flows. *WHAT* learns BoDs from ISP traces, and its performance is assessed on the benchmark traces. We consider the 500 most popular core domains seen in volunteers' browsing histories and let *WHAT* learn the BoDs using the ISP trace.

Fig. 3a shows results for the Web Browsing benchmark. The figure depicts the accuracy of *WHAT* when learning BoDs using an increasing number of flows. Training is performed using the initial part of the ISP traces – each experiment takes an increasing period of the trace for

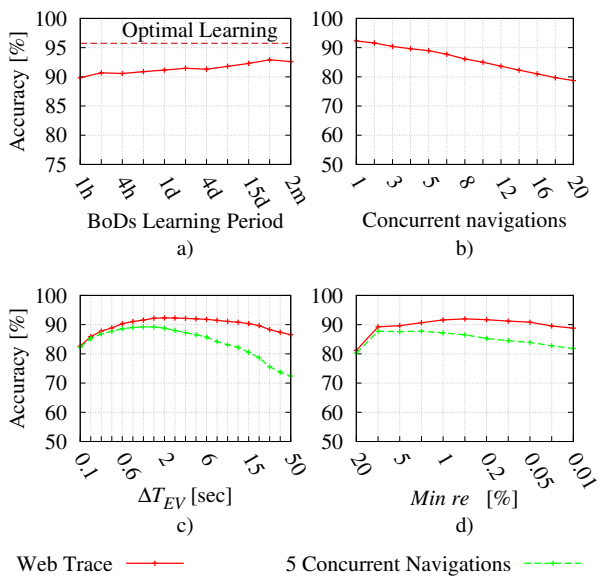


Figure 3: Accuracy vs. input data and parameters values.

learning. Accuracy in these experiments is computed as *the percentage of volume* in bytes correctly labeled on the trace.

All flows triggered by core domains not among the 500 domains considered in the training set must be labeled as "unknown" to be rightly classified. Therefore, errors occur because flows have been *(i)* labeled with wrong core domains; *(ii)* mislabeled as "unknown"; or *(iii)* labeled with core domains, while should be "unknown". Flow-wise statistics lead to similar results and are not reported.

Focusing on the left-most point in Fig. 3a, note that *WHAT* correctly classifies 90% of the traffic volume with a learning set of 1 day only. That is, most of the popular BoDs are learned by observing a single day of traffic in such medium-sized PoP/ISP. Increasing the learning set marginally improves results, with the best accuracy at around 93% with a 1-month long learning set.

The figure also reports an *"optimal learning"* line, which marks the accuracy when *WHAT* learns BoDs from the same validation trace used for testing – i.e., a biased result that gives hints on the best possible performance of the algorithm in this benchmark. By contrasting lines in the figure, we can conclude that *WHAT*, when trained with ISP traces, achieves results that are very close to the optimum one.

Fig. 3b presents the accuracy in the concurrent navigation benchmark. Results are obtained by increasing the number of concurrent users. User aggregation reduces the performance of *WHAT*. This is not a surprise, since users navigating in parallel increase the probability of ambiguous support domains appear on the trace. Overall, *WHAT* performs very close to its best accuracy when up to five users are actively browsing the web behind a single IP address. The accuracy drops to $\approx 80\%$ when more than 20 users are active. Note that the benchmark simulates users that are all active at the *same time*. This means that *WHAT* can satisfactorily operate with typical ISP traffic, where only few users are aggregated behind a home-gateway acting as a NAT.

Parameter	*Best Value*
Training set size	1 Month
ΔT_{OW}	10 Seconds
ΔT_{idle}	5 Seconds
ΔT_{EV}	[1–4] Seconds
$MinFreq$	[1–5] %

Table I: Best choices of parameters.

B. Sensitivity of Parameters

WHAT relies on a number of parameters, which affects its accuracy. We discuss Evaluation Window (ΔT_{EV}) and the minimum tf score to include domains in BoDs ($MinFreq$), since they have the highest impact on the system; for the others, the same procedure has been followed. Best choices for all parameters, including those omitted for brevity, are listed in Tab. I. For experiments in this section, *WHAT* is trained with one month of traffic from the ISP traces, and tested with the two benchmarks. In the case of concurrent navigation, we use 5 simultaneous threads.

1) **Evaluation Window (ΔT_{EV}):** Fig. 3c depicts how accuracy varies according to ΔT_{EV}. Lines represent results for the two benchmarks.

Focusing on the web browsing benchmark (red line), notice how the accuracy starts at $\approx 80\%$ when $\Delta T_{EV} = 0.1$ s, grows at the best figures (e.g., $\approx 90\%$) when $\Delta T_{EV} = 5$ s, and consistently decreases for larger values. Very small values of ΔT_{EV} cause *WHAT* to miss support domains, whereas large ΔT_{EV} values increase the chance to account for background or unrelated flows.

ΔT_{EV} becomes more important in scenarios where there are multiple navigation threads (green line). Accuracy decreases faster for large number of simultaneous navigation threads. This happens because *support domains* appearing in multiple bags can be misclassified when more than one *core domain* appears close in time.

Overall, $\Delta T_{EV} \in [1, 4]$ s provides the best trade-off.

2) ***MinFreq*** **Threshold:** The impact of $MinFreq$ is illustrated in Fig. 3d. Curves for the two benchmarks are depicted. The x-axis marks the value of the threshold – e.g., $x = 2\%$ depicts results for which any domain with tf lower than 2% in a BoD is *not* considered.

The importance of the $MinFreq$ to filter out noise from BoDs becomes clear. As an example, when $MinFreq$ is too large (e.g., 20%), domains that are popular in BoDs may be ignored, resulting in a sharp decrease on accuracy.

On the other extreme, when $MinFreq$ is low, unrelated support domains pollute BoDs. Focusing on results for $MinFreq = 0.05\%$, notice how accuracy is around 90% in the web browsing benchmark, but it is reduced to around

83% for concurrent navigation. This happens because unrelated domains in BoDs decreases classification performance.

Overall, $MinFreq \in [1, 5]\%$ provides best trade-off.

V. Early Deployment Experience

WHAT aims at accounting network usage. Monitoring only at flow level, such as with NetFlow or Tstat, guarantees a significant reduction on the volume of data exported from network monitoring equipment. However, our experience is that even flow measurements fast mount to large volumes in high-speed networks. Thus, we have implemented *WHAT* based on Apache Spark, given its easy parallel framework and built-in support for streaming processing.

To give an impression on *WHAT* performance, we benchmark *WHAT* using the ISP trace presented in Sec. III-A. Recall that the ISP trace contains more than 2.2 billion flow records. We use the full trace both for training and testing the system in a middle-sized Hadoop cluster (30 working nodes), and measure the run-time of the training and test algorithms. We find that the processing takes about 1 hour for training and 30 minutes for classification – i.e., *WHAT* can classify dozens of thousands flow records per second on each working node of the cluster. Since the problem is trivially parallelizable, we expect *WHAT* to scale well according to the cluster capacity.

VI. Related Work

A large number of classification methods are based on the inspection of packet content [1], [10], [11]. Content-based methods are however getting outdated, since encryption prevents the extraction of protocol information from network traffic. *WHAT* requires only flow-level information augmented with hostnames, which can be obtained even for encrypted protocols (e.g., using the DNS).

Behavioral techniques are also popular for traffic classification [2], [12]: The host behavior and machine-learning are used to infer protocols and applications generating traffic. *WHAT* is a behavioral classifier. Differently from previous proposals, which build models based on host addresses and port numbers, *WHAT* learns a model based on hostnames that identify flows. Thus, *WHAT* can differentiate web services even if they use the same protocol (e.g., HTTPS) and are hosted in the same infra-structure (e.g., CDNs).

The proposal in [13] is the closest to ours. It introduces a tool to identify association between flows, based on the frequency in which pairs of flows are concurrently active. *WHAT* relies on similar ideas from information retrieval to cluster flows, but it targets the classification of web traffic and operates with only flow records labeled with hostnames.

Agar et al. [14] propose to use the DNS for classification. They build a map of the whole web using DNS information. Plonka et al. [5] use DNS traffic to label flows while capturing traffic. This is used to build a classifier that separates traffic into categories. Other works (e.g., [4], [15]) share similar goals, using either DNS or SNIs found in TLS handshakes. In contrast, we address typical web services that make the majority of the traffic nowadays, ignoring well-known protocols (e.g., FTP or P2P). Moreover, *WHAT* extends such works, since it not only labels flows with hostnames, but also groups flows triggered by a single site visit. Thus, *WHAT* is able to operate even when hostnames are not informative about the initial visited website.

VII. Conclusions

This paper presented *WHAT*, describing how it mines information from flow records enriched with hostnames. Given a list of core domains representing the services to monitor, it learns the set of associated support domains contacted as a consequence. *WHAT* uses this model to categorize flows according to websites triggering the traffic. The big data approach followed by *WHAT* offers network administrators accurate per-service metering, while allowing scalability for processing large data streams.

References

[1] A. Callado *et al.*, "A Survey on Internet Traffic Identification," *Commun. Surveys Tuts.*, vol. 11, no. 3, pp. 37–52, 2009.

[2] H. Kim *et al.*, "Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices," in *Proc. of the CoNEXT*, 2008, pp. 1–12.

[3] D. Naylor *et al.*, "The Cost of the "S" in HTTPS," in *Proc. of the CoNEXT*, 2014, pp. 133–140.

[4] I. Bermudez *et al.*, "DNS to the Rescue: Discerning Content and Services in a Tangled Web," in *Proc. of the IMC*, 2012, pp. 413–426.

[5] D. Plonka and P. Barford, "Flexible Traffic and Host Profiling via DNS Rendezvous," in *Proc. of the SATIN*, 2011, pp. 1–8.

[6] "Common Crawl," http://commoncrawl.org/.

[7] A. Finamore *et al.*, "Experiences of Internet Traffic Monitoring with Tstat," *IEEE Netw.*, vol. 25, no. 3, pp. 8–14, 2011.

[8] K. S. Jones, "A Statistical Interpretation of Term Specificity and Its Application in Retrieval," *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[9] "Selenium," http://www.seleniumhq.org/.

[10] T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification Using Machine Learning," *Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 2008.

[11] H. Yao *et al.*, "SAMPLES: Self Adaptive Mining of Persistent LExical Snippets for Classifying Mobile Application Traffic," in *Proc. of the MobiCom*, 2015, pp. 439–451.

[12] T. Karagiannis *et al.*, "BLINC: Multilevel Traffic Classification in the Dark," in *Proc. of the SIGCOMM*, 2005, pp. 229–240.

[13] S. Kandula *et al.*, "What's Going on?: Learning Communication Rules in Edge Networks," in *Proc. of the SIGCOMM*, 2008, pp. 87–98.

[14] B. Ager *et al.*, "Web Content Cartography," in *Proc. of the IMC*, 2011, pp. 585–600.

[15] P. Foremski *et al.*, "DNS-Class: Immediate Classification of IP Flows using DNS," *Int. J. Netw. Manag.*, vol. 24, no. 4, pp. 272–288, 2014.