4-18-2014

Detection, Classification, and Workload Analysis of Web Robots

Derek Doran
University of Connecticut - Storrs, ddoran.home@gmail.com

Detection, Classification, and Workload Analysis of Web Robots

Derek Doran, Ph.D.

University of Connecticut, 2014

It has been traditionally believed that humans, who exhibit well-studied behaviors and statistical regularities in their traffic, primarily generate the stream of traffic seen by Web servers. Over the past decade, however, the Web has seen a drastic increase in the number of requests initiated by contemporary Web robots or crawlers. These robots, whose traffic can be significant (upwards of 45% on the UConn School of Engineering Web server and 70% across digital libraries), exhibit sophisticated functionality and have widely varying demands. To prepare Web servers to handle this new generation of traffic with high performance, to develop methods that control and limit their behavior, and to understand how they interact with the social and sensitive data shared on the Web, a deep understanding of Web robots and their traffic qualities is essential. Unfortunately, the current understanding of robot traffic and their impact on the performance of a Web server is minimal. This deficiency is compounded by the fact that: (i) state-of-the-art methods for identifying their visits are very limited; and (ii) owing

to the fundamental behavioral differences between robots and humans, we cannot assume that our knowledge of human behavior and traffic features transcend to robots. This dissertation addresses the above deficiencies and carries out a comprehensive evaluation of Web robot traffic on the Internet. We first introduce and demonstrate the effectiveness of a new approach for detecting robot traffic that is rooted in fundamental differences between robot and human behavior, and can run offline or in real-time. Secondly, we propose a multi-dimensional classification scheme to decompose robots based on their functionality, resource favoritism, and workload demands. Thirdly, using traces of requests to Web servers across many Internet domains, we reveal critical differences in the way robot and human traffic qualities do (not) exhibit power-tailed trends and long-range dependence in their arrival processes using a suite of analysis tools. Finally, we propose a novel predictive caching algorithm that can service Web robot and human traffic simultaneously and with much higher performance compared to caching algorithms that are used in practice.

Detection, Classification, and Workload Analysis of Web Robots

Derek Doran

M.S., University of Connecticut, 2012

B.S., University of Connecticut, 2007

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2014

APPROVAL PAGE

Doctor of Philosophy Dissertation

Detection, Classification, and Workload Analysis of Web Robots

Presented by

Derek Doran, B.S., M.S.

Major Advisor

Swapna Gokhale

Associate Advisor

Lester Lipsky

Associate Advisor

Jun-Hong Cui

University of Connecticut

2014

ACKNOWLEDGEMENTS

I want to thank my advisor Swapna Gokhale for her incredible guidance and support. Swapna gave me the freedom to study interesting problems from the start, charged me with devising my own research questions, and cared so much about teaching those things that successful academic CS researchers should but may never tell you. Her advocacy opened doors that led to incredible research experiences across the country and the world. She forged my writing style and critical thinking abilities, and capacity for patience thanks to our sometimes endless paper revision cycles, which in hindsight I really appreciate. Above all, she let me run with my own ideas, and believed in me.

Thanks also to my great set of mentors - Drs. Lester Lipsky, Aldo Dagnino, and Veena Mendiratta. I'll fondly remember the one-on-one lectures in Prof. Lipsky's office and home, the interesting problems we discussed together, and his constant encouragement. The discussions I had with Aldo in his office at ABB in North Carolina was the highlight of my year-long position in his research group. He was more than happy to share his unbiased thoughts about industry and academia and gave excellent advice about being a successful researcher. His wisdom and perseverance (as a researcher, manager, and Ironman competitor) has left a lasting impression on me. Veena showed complete trust in my research

and technical abilities, and while I worked with her at Bell Labs for two summers, she also gave me the leeway to follow my own ideas. She selflessly allowed me to engage in multiple papers and patents with her, and even invited me to work on her own projects outside of Bell Labs. Her lessons about working in industry and constant encouragement pushed me to become the researcher I am today.

Thanks also to Rohit Mehta from Engineering Computer Services at the University of Connecticut School of Engineering, Marc Maynard and Darlene Hart from the Roper Center for Public Opinion Research, and Michael Jean from the University of Connecticut Coop for making their Web server logs available for this dissertation.

Finally, thanks to my parents for supporting me while I did my grad school thing, and to all of my friends who kept me together and didn't make fun of me (that much) for not having a 'real job' for seven years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction and Motivation

A Web robot may be generally defined as an autonomous program that sends requests to servers across the Internet requesting some resources. The program then analyzes the results received, or sends the results to a central agency, to extract knowledge from the collected data to fulfill some specific purpose in a broader context. A canonical example of a Web robot is a search engine indexer, while a less common example is an RSS feed crawler or a robot designed to collect Web sites for an Internet archive. Over the past decade, the proportion of http traffic that can be attributed to these robots have risen at a staggering rate [86,94,35,60]. Whereas studies at the turn of the century identified approximately one out of ten requests to be generated by Web robots [86,94], recent measurements find that robot traffic constitutes at least half of all the traffic seen by academic, e-commerce, testing, and vertical search engine sites [109].

This rapid rise in Web robot traffic may be attributed to the proliferation of online social media services which promote user-generated content that has lead to a dramatic increase in the volume of time-sensitive, dynamic information

posted to the Web. Many organizations perceive value in such user-generated content because it provides significant insights into users' opinions about products, services, and their emotions and reactions to current events [59]. posted to the Internet. As a result, to keep data repositories up-to-date, contemporary Web robots need to be more comprehensive in their searches, more specialized in their functionality (to analyze a wider variety of data) [31,33], more frequent in their visits (to stay abreast on content that changes dynamically and rapidly) [60,30], and employ advanced algorithms for comprehensive crawls [5,97]. For example, modern Web robots are capable of crawling news, social networking sites, and user blogs to harvest emotional thoughts and feelings [68].

Presently, robots can crawl Web servers in an unregulated and wild manner. They may behave unethically by ignoring the conditions imposed by administrators in the file *robots.txt* [75]. They can also violate the privacy of users by collecting their personal information posted on Web sites [45]. Organizations may also be tempted to employ aggressive robots with questionable behaviors for harvesting personal data that users innocently share via popular, public online social network and social media services. This unrestricted behavior may not only compromise user privacy, but may also impose onerous demands on Web servers [32]. Furthermore, because robot traffic differs statistically from conventional Web traffic [38,35], many Web server optimizations may not perform adequately, and further drag down server performance.

The properties of Web traffic have been studied extensively over the past two decades [7,57,77]. These studies were based on Web access logs that primarily consisted of requests from humans. Intuitively, however, the autonomous way in which Web robots send requests to retrieve information off a site stands in sharp contrast to the way humans browse a Web site. For example, a robot may send requests at a constant request rate to retrieve all the resources at a site, and then perform the processing to extract knowledge after the fact. Human-induced traffic, however, is much more deliberate. A human commonly visits a site with the goal of finding specific knowledge, and then analyzes the structure and data on the existing pages to help find the information desired. Furthermore, the traffic properties of humans are also induced by the characteristics of a Web browser. For example, Web browsers may quickly send requests for all embedded resources on a html page causing human traffic to be bursty. Due to these differences, our current body of knowledge about Web traffic may not transcend to Web robots.

Although a small number of previous studies have attempted to better our understanding of Web robot traffic, they are inadequate for three important reasons. First, state-of-the-art Web robot detection techniques are unable to identify contemporary robots that exhibit many different functionalities, and whose behavior evolves over time [36]. Second, existing studies about robot traffic do not consider the varying behavior, functionality, or favored resources of individual robots, even though evidence suggests that different robots exhibit very different

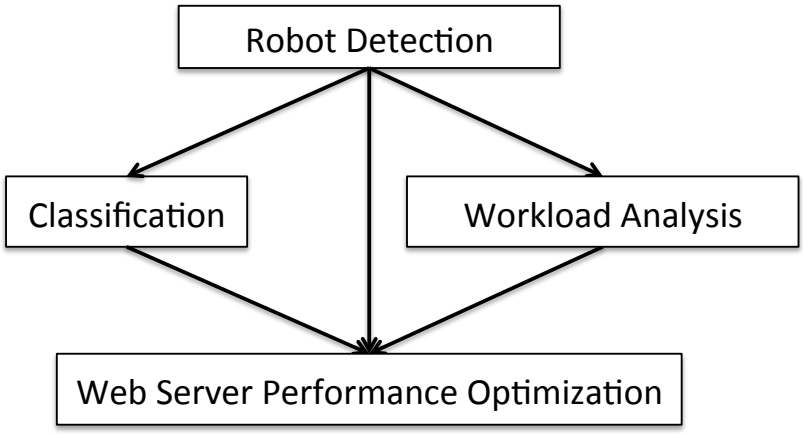


Fig. 1.1: Relationship among the studies in this dissertation

qualities [31]. Finally, existing characterization studies only provide summary information about robot traffic, including average bytes transferred, total number of requests, and the size of http responses [30,63].

The objective of this dissertation is to perform a comprehensive evaluation of Web robot traffic on the Internet, which we carry out in four studies outlined in Figure 1.1. In the first step, we enable a comprehensive evaluation by developing algorithms that can detect Web robot traffic within both Web server logs and within real-time streams of Web traffic. In the second step, we propose a multi-dimensional classification scheme for Web robot traffic, dividing their traffic according to their intended functionality, traffic characteristics, and types of resources they request during their sessions. Through the lens of this multi-dimensional view, we identify patterns and behaviors that differentiate the many types of Web robots that crawl the Web. In the third step, we introduce new

tools for performing detailed analysis of heavy- and power-tailed features of Web traffic, and apply them to robot sessions. Through this application, we identify extremely important statistical contrasts between Web robot and human requests. These contrsts have serious implications on the performance of Web servers that are optimized to handle specific statistical trends in its stream of requests. In the final step of our evaluation, we integrate the lessons learned from the previous three steps to develop a novel predictive caching scheme that can alleviate a significant amount of the strain that robot requests impose on a Web server.

The dissertation is organized as follows. Chapter 2 offers a detailed survey of past Web robot detection algorithms and, identifies their limitations, and proposes state-of-the-art methods for offline and real-time detection. Chapter 3 introduces a multi-dimensional scheme for Web robot traffic, used to identify new patterns in Web robot behavior. Chapter 4 assess whether or not important features of Web robot traffic take on heavy- or power-tailed distributions across three different Web servers from different domains of the Internet, and discusses how these distributions differ between robots and humans. Chapter 5 combines our offline and real-time robot detection method and our findings from our classification scheme and traffic study to develop a new predictive caching architecture and policy to service robot and human traffic. We find that our approach outperforms a suite of baseline methods by a large margin. Chapter 6 summarizes the contributions of this dissertation and gives avenues for future work.

Chapter 2

Web Robot Detection

A prerequisite for any study about robot traffic is a mechanism that can reliably identify Web robot sessions. This robot detection typically occurs during an *offline* or post-mortem analysis of Web server access logs. Offline detection is necessary to develop a deep understanding of how robots crawl, their traffic characteristics, and functionality. Post-mortem analysis can also be used to enhance server security by discovering resources that robots favor and where these are being sent based on an IP address or user-agent look up service [110]. This analysis can thus suggest whether a Web site may benefit from more secure or strict data publication policies and access control methods. Offline detection, however, can only identify robots in retrospect or after-the-fact. While retrospective detection can be used to guide the development of proactive strategies to limit the damage caused by unethical or offensive robots, it does not offer Web servers the ability to distinguish between active robot and human sessions. Instead, distinguishing between robot and human sessions in *real-time* is necessary to limit the impact of robot traffic on a Web server, by providing them differentiated treatment depending on their

behaviors and on the present server conditions. For example, when a Web server is under heavy load, robot requests can be handled with a lower priority. Robot traffic may also be redirected to specialized caches [105] so as to not disrupt resources in ones tailored for human traffic. Administrators may also deny robots' requests for specific types of files; for example pdf or doc files on corporate Web servers, out of concern that other organizations may be using these agents to harvest critical information. As another example, robot sessions that successfully enter secure areas of an online journal or a news site can be curtailed before they begin to collect any premium, pay-to-access content. It is thus clear that offline and online detection, each offer distinct, unique advantages, and hence, both these approaches must be employed synergistically to protect Web servers from ill-behaved, malicious robots.

In the early approaches to discover robots from Web logs, the user-agent field of each http request in the logs was compared against a database of regular expressions. The contents of this database, however, cannot be updated at the same pace as the rapid growth in the novelty and sophistication of Web robots. As a result, referencing a static database can only be useful in detecting old and common Web robots. Thus, while this method may be simple to implement and sufficient to detect common robots, it may not be as effective in detecting new and evolving robots. A limited number of research efforts over the past decade have refined these simple detection approaches, enriching the ability to detect

a broader range of robots. These techniques vary widely in their underpinning detection philosophy. For example, some consider different fields in the Web log or identify simple patterns in sessions, while others perform a statistical analysis across different features of Web robot traffic or use machine learning models for detection. Some techniques even implement an entire system to detect robots in real time, rather than by mining features in the Web server access logs. While each of these techniques has been effective in its limited experimental demonstration, their critical comparison either via experimentation or by a reasoning of their features and capabilities is lacking. We argue that in order to develop robust robot detection approaches that will continue to remain effective as the robot traffic evolves, it is first necessary to understand these contemporary techniques and identify their strengths, weaknesses, and differences.

In this chapter, we survey the existing approaches to detect Web robots and propose a classification scheme that categorizes the contemporary detection techniques in Section 2.1. The intention of our classification scheme is to exploit the commonalities across the different techniques, highlight their differences, and evaluate the 'types' of techniques that will be most effective for detecting modern Web robot traffic. Based on this comparative analysis, Section 2.2 introduces a new, integrated method that supports both offline and real-time Web robot detection. Our detection approach relies on the training of analytical models over the request patterns of robots and humans for different types of resources. We

motivate why the resource request patterns of robots and humans are intrinsic to their respective behaviors, and hence, are fundamentally distinct from each other. We also describe how the contrasts in these request patterns are likely to be immutable and persistent, even if robots and humans change their behaviors over time. We evaluate the performance of our methodology for both offline and real-time detection by playing back multiple streams of robot and human sessions collected from the University of Connecticut (UConn) School of Engineering (SoE) Web server. The results indicate that our approach can separate robots from humans with high accuracy for varying session lengths and proportions of robot traffic. Our approach thus offers two key advantages over contemporary detection techniques, namely, it: (i) defines a model based on a fundamental, time-invariant difference between robot and human behavior; and (ii) can be simultaneously used for highly accurate offline and real-time detection[1].

2.1 Survey and Analysis of Contemporary Techniques

We define the problem of identifying robot traffic on a server as the Web robot detection problem, and formalize it as follows: Given a set of http request records R, for each record $r \in$ R classify r with a label specifying whether the request was sent by a human user or sent by some automated program or a system. Examples of such programs or systems include Web site indexers, spambots, and

[1] Portions of this chapter were previously written and published by the author in [36,34].

utility programs that scan Web sites for security vulnerabilities. Each r follows a format defined by the Web server administrator, depending on how verbose or simple they wish the server access log to be. Traditionally, r consists of at least the address from which the request originated, the location and the name of the resource requested, the http response code (i.e., 404 if the file is not found on the server), and the user-agent field from the request packet.

Many proposed solutions to the robot detection problem classify *sessions* rather than individual records. A session $\mathbf{S} \subseteq \mathrm{R}$ is a collection of all request records from a user during a single visit. In this case, the label assigned to $\mathbf{S}$ is assigned to each $r \in \mathbf{S}$ implicity, so that all $r \in \mathrm{R}$ are classified. For techniques that characterize sessions instead of individual requests, it is also necessary to identify such sessions from the collection of records. We formalize the session identification problem as follows: Given a set of http records R, find the set of sessions $\mathcal{S} = \{\mathbf{S_1}, ..., \mathbf{S_n}\}$ such that the sessions of $\mathcal{S}$ are mutually exclusive and $\bigcup_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{S_i} = \mathrm{R}$.

Such real-time detection techniques do not operate over Web server logs, making the above problem formulation inappropriate for these techniques. They instead solve a variant called the *real-time robot detection problem*, which we formulate as follows: given a set of active sessions on a Web server $\mathcal{S}_\mathcal{A}$, determine whether the traffic in each session $\mathbf{S} \in \mathcal{S}_\mathcal{A}$ is produced by a human or by a robot before the session $\mathbf{S}$ is terminated. This variant analyzes an incoming stream of

requests to determine in real time whether an active session is human or robot-induced. Naturally, real-time robot detection is more difficult than detection based on server access logs as the data used for detection is limited only to observations from the current session.

The real-time robot detection problem may be thought of as a combination of both *intrusion prevention* and *intrusion detection* problems. These are related approaches that attempt to prevent unauthorized users from accessing a given system (the prevention problem) or detecting that unauthorized users are currently using the system (the detection problem). The main advantage of real-time robot detection is that it makes it possible to impose immediate restrictions on the robots' access to site resources. Such restrictions might involve blocking the robots, preventing them from accessing specific resources, or servicing their requests with low priority compared to human sessions when the server is under excessive load. The margin of error in real-time detection must be very low, however, to minimize the risk of alienating a human because of an incorrect classification and the subsequent imposition of restrictions. Real-time robot detection is a hybrid of both intrusion prevention and detection aimed specifically at Web robots, where their visits may be detected before or during an active crawl of the site.

Solutions to each of the above robot detection problems are important in different contexts. Real-time detection is most important when the security of a

site is of paramount concern. As already alluded to, sensing what active sessions are from robots can help halt these robots from continuing to browse a site and from accessing sensitive materials. In addition, real-time detection could be used to block robot attacks by terminating their sessions before a server-side exploit or DDoS attack causes any damage. Solutions to the offline detection problem are useful to filter robot traffic for a characterization study that considers only humans or vice versa.

2.1.1 Hierarchical classification

We classify the existing robot detection techniques into four categories, according to their underpinning analysis or detection philosophy. We first describe the four categories and then summarize the techniques belonging to each one of them. The first three categories include techniques that solve the offline detection problem, while the fourth category is comprised of techniques that solve the real-time detection problem. The categories are presented in the order of their detection "strength". We define the strength of a detection category as its potential to detect previously unknown but well-behaved robots or sophisticated robots designed to evade detection. In practice, the selection of a specific type of technique used should balance the complexity and feasibility of its implementation against protecting the security of the site. For example, if the goal of detection is to extract a sample of Web robots from a Web log or to filter common, high-demand

search engine Web robots such as *Googlebot* or *MSNBot* from an access log before running usage reports, a weak detection technique that is simple to implement may be appropriate. If the goal of detection, however, is to block robots that mask their visits, or to identify specific types of robots that may pose a security risk to a site, a stronger technique may be suitable. The four types of Web robot detection techniques are as follows:

- **Syntactical log analysis:** Syntactical log analysis techniques attempt to discover robots through a simple processing of access logs. This processing involves looking for keywords in user-agent fields of robot requests or reviewing the host addresses to identify locations from where robots are known to originate. Some of the earlier detection techniques were based on syntactical log analysis of server access files. The primary appeal of these techniques lies in the simplicity of their implementation and their reliance on a readily available file. Unfortunately, such techniques can only find those robots that are previously known, because they are based on a prior knowledge of specific key words and addresses from which robots originate.
- **Traffic pattern analysis:** Traffic-based analysis techniques seek to find common characteristics of Web robot traffic that contrast with the features of human traffic. For example, a traffic analysis technique may rely on discovering conventional robot navigational patterns that involve a DFS or a BFS search of all embedded links and resources from a main, home, or

an index page of a Web site. Another technique may analyze the types of resources that robots request to discover patterns indicative of robot behavior, or consider the statistical contrasts between robot inter-arrival times or session length distributions with those of a human visitor. The key feature of traffic pattern analysis techniques is that they attempt to detect robots based on fixed expectations about their behavior rather than using the knowledge of their names and origins as in the case of syntactical processing techniques. Thus, we consider traffic analysis based detection as stronger than detection based on syntactical log analysis.

- **Analytical learning techniques:** The random nature of robot visits and Web server traffic naturally lead to formal probabilistic models as a way to detect robots. These techniques exploit the observed characteristics of the logged sessions to estimate the likelihood that a given session was generated by a robot using a formal machine learning or a probabilistic model. The sessions may be characterized using several metrics such as time between requests, session length, html-to-image request ratio, and percentage of requests made using the http HEAD command. Analytical learning techniques can also be retrained over time as the statistical trends in Web robot traffic evolve. Thus, we consider analytical learning techniques to be even stronger than detection based on traffic pattern analysis. The primary disadvantage of analytical learning techniques, however, is that they require a training

data set that includes robots that are difficult to detect in order to ensure reasonable detection accuracy.

- **Turing test systems:** Analogous to how a Turing test tries to reliably classify a conversation as being produced by a computer or a human in real time [116], the robot detection problem tries to determine if server sessions are being produced by a robot or a human, where the http request/response pairs within a session is like a dialogue. We classify a technique that issues a test to active sessions, analyzes the response, and then classifies the session based on the results as a Turing test system. Such systems are designed to perform real-time Web robot detection, a type of intrusion detection problem. These tests may either present a CAPTCHA [3] challenge to the visitor or engineer Web sites such that the occurrence or non occurrence of a certain event can help to determine whether the session passes or fails the Turing test. While these techniques have a theoretical and algorithmic underpinning, they are best recognized by their use of creative engineering techniques to force active sessions to take the Turing test. In the context of Web robot detection, active sessions that fail the Turing test can immediately be classified as Web robots.

Figure 2.1 visualizes our classification scheme for Web robot detection techniques. As previously discussed, the techniques may be first classified according to whether they perform offline or online detection. Online detection techniques

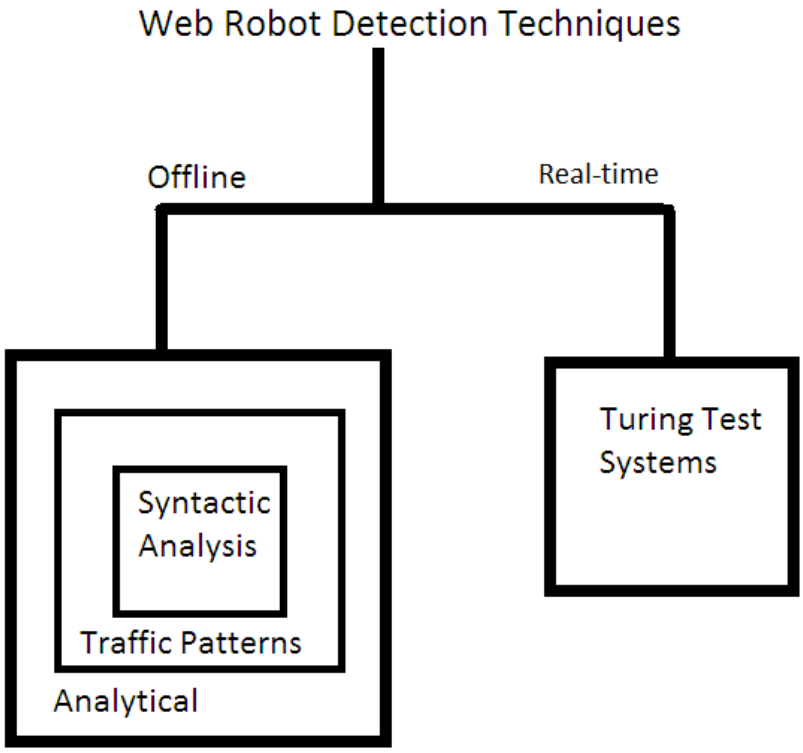


Fig. 2.1: Web robot detection methods hierarchy

are labeled as Turing test systems. Offline detection techniques may be further refined into syntactical log analysis, traffic pattern analysis, and analytical learning models. The nested structure in Figure 2.1 reflects the increasing strength of each type of technique in the offline detection category. Techniques that fall into a nested classification are more limited in the types of Web robots that they may detect. Table 1 summarizes the major techniques that belong to each category that this survey covers. Next, we discuss each of these techniques in detail.

2.1.2 Syntactic log analysis

Syntactic log processing techniques analyze each entry in the server access log based on the information recorded in the log. The first technique in this category considers only individual fields in the logged request. The second technique references a database of known robot user-agent fields, and subsequently compares each request against this database. The final technique identifies robots based on multiple facets recorded in a Web server log.

Detection through individual field parsing

The most simple syntactic detection techniques commonly used in many commercial and open-source systems, such as *AWStats* [9], rely on comparing individual fields of each request against a database or a list of keywords that indicate that the request originates from a Web robot. While very simple to implement and understand, robots can easily circumvent detection by setting such fields to some arbitrary strings or by copying the fields for some well-known user agents. This is a rampant problem - in our study of Web robot activity from the UConn SoE Web server, *AWStats* detected a disproportionate volume of http requests from the three most popular robots on the Internet – *Yahoo Slurp*, *MSNBot*, and *Googlebot* [31]. We believe that such a weak technique is still used in popular commercial systems because most administrators still do not consider robot traffic as a significant cause for concern despite evidence to the contrary.

Detection through user-agent mapping

Kabe and Miyazaki [67] classify clients based on the specific application being used, rather than classifying the sessions as human- or robot-induced. They argue that learning the specific application type is important to ensure that a Web page is properly presented and to protect the Web site from abuse. Their classification system partitions the types of user agents into browsers, indexing robots, offline browsers, link checkers (a type of robot), update detectors (applications that poll a site to notify a user if there is an update), accelerators (pre-fetchers that get a target of a hyperlink contained in the current document before the user selects it), cache servers, and BBS Autopilot (a tool that checks and posts information or spams advertising on Web forms).

The analysis is applied to the access logs using a two-stage approach. In the first stage, the user-agent strings retrieved from the access logs are classified according to the product. Similar user-agent strings that resemble the same product (i.e., Mozillia 4.0, MSIE/5, etc.) are aggregated into a single product name using pre-defined rules. In the second stage, rules are developed to map each product name to a user-agent characteristic (Browser, Offline Browser, Indexing Robot, etc.). For example, the user-agent string `Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; MathPlayer 2.1; .NET CLR 1.1.4322)` represents a visit by Internet Explorer 6.0 with MathPlayer 2.1 installed. In stage one, this user-agent field would be aggregated into the Internet Explorer product name. In stage

two, the Internet Explorer product name would be aggregated into the Browser characteristic class.

The results indicate that the technique detects 2% more offline browsers (a type of harvester robot) that would be undetected by a weaker http-based method (using first token of the user-agent string). This work was among the earliest that gave a scheme to detect unique types of robots, such as spam harvesters and logo trackers [96].

Detection through multifaceted log analysis

Huntington *et. al.* proposed a multi-step log analysis technique which utilizes syntactic parsing to detect robots that visit the online scientific journal *Glycobiology* [60]. In the first step, the IP addresses that sent a request for *robots.txt* were identified and immediately classified as robots. The robots identified in this step accounted for 0.5% of all traffic. Next, a reverse DNS lookup was performed for each IP address in the log file. The DNS names that included the terms robot, bot, search, spider, and crawler were checked by a human expert to be manually classified as Web robots. The robots identified by the reverse DNS lookup accounted for 16.1% of all traffic. Finally, the authors used a database of IP addresses of known robots, that do not declare themselves in their user-agent fields. This IP address database was obtained from the Web. The robots identified using this IP address database accounted for an additional 6.5% of traffic. In total, they found

that their multi-faced syntactic log analysis identified robots that accounted for almost a third (32.6%) of all traffic. The percentage of robots at this site may be high because only a small number of human users are interested in visiting an online scientific journal, whereas archival or scholarly article search services will commonly employ robots to visit the journal frequently to index or archive new articles.

2.1.3 Traffic pattern analysis

Detection techniques which analyze patterns in robot traffic use a deeper interpretation of the entries in the Web server access log rather than superficially focusing on the contents of their user-agent fields. They consider traffic characteristics such as the types of resources requested and attributes of requests such as the volume, referring location, percentage of errors, and time-of-day. In establishing these detection techniques, the authors first study Web robot traffic to identify distinguishing characteristics and statistical patterns. The detection technique is then formulated to automatically discover these distinguishing features in order to classify clients as humans or Web robots.

Detection through syntactic and pattern analysis

Geens *et. al.* propose a detection technique which combines syntactic analysis with traffic pattern analysis [44]. They first study the detection performance of

Approach	Correct	Incorrect	Recall	Precision
manual	241	0	1	1
robots.txt	41	0	.1701	1
ip address	167	0	.6929	.9940
user-agent	64	0	.2656	1

Table 2.1: Performance of individual field parsing approaches

several different syntactic parsing methods individually. The first parsing technique considers requests to the file `robots.txt`. This file is defined in the Robot Exclusion Standard [74], that proposes a protocol where Web robots request `robots.txt` when they first visit a site. They reason that this is an unreliable approach to detect robots because this protocol is voluntary, and hence unenforceable. Next, they compare the IP address field of the request against a list of addresses from which Web robots are known to originate. The large number of robots, combined with the presence of proxy servers and dynamic IP addresses, lead them to conclude that this also is an unreliable approach to detect robots. Finally, they compare the user-agent field against the entries in a database of known regular expressions.

These individual parsing techniques were evaluated on the set of 241 Web robots that were identified manually from a Web access log. The performance of each technique was determined using the metrics *recall* (r) and *precision* (p) defined as:

$$r = \frac{|\mathbf{S}_r^c|}{|\mathbf{S}_r|}$$

$$p = \frac{|\mathbf{S}_r^c|}{|\mathbf{S}_r^p|}$$

where $\mathbf{S}_r$ is the set of all robot sessions, $\mathbf{S}_r^p$ is set of all sessions labeled as a robot by the detector, and $\mathbf{S}_r^c \subseteq \mathbf{S}_r^p$ is the set of all sessions labeled correctly. r measures the percentage of robots captured while p measures the *precision*, or percentage of sessions correctly classified as robot induced. Table 2.1 summarizes the results, which suggests that each individual field parsing techniques lead to very few false positives at the cost of missing a significant number of Web robots.

In order to improve r, the percentage of robots detected, they also consider three simple patterns in their traffic. The first identifies requests that use the http HEAD command to retrieve only the headers of the http responses rather than the entire response. They choose this feature because they reason that some Web robots will use the HEAD command if they are only interested in checking whether a certain resource exists, such as link checkers, while humans using a Web browser will use the http GET command to retrieve an entire html page from the server. The next pattern checks whether all requests in a session have an unassigned referrer field. Most robots choose to leave the referring field blank, while Web browsers often assign the previous page visited to the referrer field. Finally, the authors examine whether any image files are requested during a session. They consider image files because most robots are not interested in the embedded images on html pages.

This combined approach leads to the following rule to detect Web robots: *if*

a session contains a request for `robots.txt` *OR its IP address is in the robot IP list OR its user-agent field is in the robot user-agent list OR the HEAD method is used OR (the referring field is unassigned AND no images are requested) then classify the session as a Web robot.* This rule obtains a recall value of 0.9731 with precision 0.8935 over the same Web log consisting of 241 Web robots. Comparing these results against those in Table 2.1, it can be seen that the combined approach is superior to using only a single syntactic parsing technique. This study thus highlights how a traffic pattern analysis approach is stronger than performing only syntactic analysis.

Detection based on resource request patterns

Guo *et. al.* exploit the assumption that robots request only certain resources when they traverse a site [51]. Hence, they try to detect Web robots based on the patterns in their requested URL resources, giving rise to two new detection algorithms. The first algorithm considers the volume while the second one considers the rate of resource requests.

The first algorithm involves a classification of requested URL resources. The resources are divided into 8 different types, namely Web page, document, script, image, music, video, download, and other. In the first step, entries in the log file are grouped according to the type of resource requested. In the second step, the host and user-agent fields are parsed from each group. For each group, records that

carry the same address and user-agent field are treated as a collection of requests coming from a single visitor. If the time interval between two requests from the same visitor is longer than a fixed threshold, then the requests are considered to be from different sessions. All sessions in all groups are formed using this method. In the third step, sessions are marked as a "robot candidate" if a single resource type was requested. In the fourth step, the first three bytes of the IP address and user-agent fields are checked for each "robot candidate" session. Those that match are treated as referring to a single robot. Finally, the number of sessions of the same type generated by the same robot candidate and the number of corresponding visiting records are counted. This data is collected to recognize sessions that only request a single main type of resource, although these may be of significant length. It is possible, however, that humans may also request only one type of resource during their visit, if for example they know the specific location of a resource and navigate to it directly. To address this, the algorithm also considers the number of visits recorded for that client to distinguish robots that make repeated visits against humans that visit the server only once to obtain some resource. If the count of the number of sessions of the same type generated by the same robot candidate and the number of visits recorded both exceed pre-defined thresholds, the session is classified as being from a robot.

The second algorithm considers the rate at which resources are requested as well as the *member list of a Web page*, defined as the aggregation of all URLs

of the embedded objects (e.g., images, frames, sounds, etc.) in the page. The idea is to generate a list of the requested resources and the times at which they were requested. If the time difference of all the requests in a list is greater than some threshold, then the visitor should be classified as a robot. Furthermore, if a Web page is accessed but not all embedded objects are requested (e.g. only a subset of a *member list* is requested) then the user can also be suspected to be a robot. The algorithm assumes that the request patterns of humans are governed by the behavior of a common Web browser. Thus, an initial request for an html Web page is followed by a barrage of requests for the embedded resources sent by the browser as it renders the site in real time. Thus, if all the resources are not requested, then the visitor is less likely to be a human.

The detection capabilities of the two algorithms are compared over the same set of data. Their experimental results showed that their algorithms detect all robots that exhibit good behavior (which they define as a robot that requests the file *robots.txt*). In terms of accuracy, of the 253 clients marked as possible robots, 28 clients were positively identified by setting threshold values for the number of sessions of the same type generated by a robot at ≥ 2 and the number of corresponding recorded vists at ≥ 5. They set these thresholds based on the assumption that robots divide their work into several subtasks, and hence, produce more sessions than a human user. Also, they assume that robots request a "bit more" content than a human. Of the 28 clients, 20 were "well behaved" while 8

would have remained undetected as they did not request *robots.txt*.

Detection through query rate patterns

Duskin *et. al.* separate robots from humans in Web search engine logs by analyzing the rate at which search queries are sent [41]. They argue that traditional detection techniques are inapplicable to access logs recorded by Web search engines because the logs do not record all the fields in the http request packet that a Web server access log does. Furthermore, the standard approach for detecting robot requests in a search engine access log, based on setting thresholds off of various metrics [62], are inadequate. While such a heuristic technique may be useful to filter a majority of Web robots, it is incapable of detecting those that do not send search queries at a rapid rate.

In order to improve upon this heuristic approach, the authors propose that the activity patterns of search users must be studied using multiple metrics, such as the rate of query submission, time interval between queries, rate at which users type, duration of sessions of continuous activity, correlation with time of day, and the regularity of the submitted queries. In this work, the authors examine the behavior using the average rate of queries submitted and the time interval between successive queries. The philosophy of the methodology is to first set thresholds on one attribute in order to partition the sessions into robots and humans, and then subsequently examine the differences in the distribution of the

other attribute for the members in the robot and human classes to identify the critical distinctions between the two. Thus, their paper introduces two separate robot detection techniques.

Three data sources are considered: logs from AllTheWeb in 2001 (ATW), AltaVista in 2002 (AV), and MSN in 2006. First, they analyze each log by classifying sessions into Web robots and humans based on the number of queries per session. For the AV and MSN datasets, they classify sessions with less than 10 queries as being a human, and the rest as Web robots. For the ATW dataset, they classify sessions with less than 50 queries as human and sessions with greater than 300 sessions as Web robots. Different thresholds are used for the initial classification because ATW features click-through data, so it is expected that there will be more entries per user relative to the MSN and AV datasets. They also distinguish between different queries and resubmission of the same query many times, which may occur when a user clicks on links from the result page and then returns to the results, or requests additional search results.

Next, the authors classify Web robot and human sessions based on the smallest interval between different queries in a session. They assume that humans cannot submit a new query within one second of the previous query, thus sessions containing an interval of less than one second are considered to be from robots. Sessions whose smallest time between queries is greater than 25 seconds in the ATW and MSN datasets and greater than 10 seconds in the AV dataset are

classified as humans. The threshold for the AV dataset was lower because of the shorter delays observed in that log. From this classification, the distributions of the number of queries submitted by Web robots and humans were studied. The results show that the probability of a human exhibiting a given number of queries per session decreases monotonically with the number of queries. It does not monotonically decrease, however, for Web robots.

Detection using traffic metrics

Lin *et. al.* introduce a scheme to categorize user sessions into different groups [79]. This study postulates that modern Web traffic is multi-class, consisting of humans, Web robots, and other Internet protocols, such as peer-to-peer file sharing. They propose a series of metrics to measure session properties and then use these metrics to heuristically determine whether a session is generated by a human, Web robot, or is P2SP (Peer-to-Server-Peer) traffic. To develop their proposed metrics, they first assume that "short" sessions (defined as sessions with only a single request) are generated from "gentle crawlers" (a type of Web robot) or P2SP clients, very long sessions are always generated by a Web robot, and that a Web robot session is likely to send requests for a larger variety of resources when compared to human or P2SP clients that target specific information on a Web server. They also assume that Web robots are most interested in html, htm, jsp, and asp files and favor them over all other types of resources.

They define the metric *Human Similarity* (HS) in order to estimate the commonality between sessions generated by humans and other classes of users. The HS of a session is defined as:

$$HS = \sum_{i=1}^{n} s(C_i)w(S_i)$$

where $s(C_i)$ is a score assigned to the response code of the i^{th} request C_i and $w(S_i)$ is the weight of the type of resource requested during the i^{th} request S_i. If S_i corresponds to a file type commonly requested by P2SP and other Web robots, such as mp3, wmv, and exe files, the weight is assigned as $S_i = 10$, otherwise $S_i = 1$. The response code scores are negative for 4xx and 3xx responses, positive for 206 and 200 responses, and 0 for all other codes. No explanation is provided for the assignment of these values. They observe that the probability density function of HS contains four peaks; two that appear near zero on both sides, and two that represent an extremely high and low HS respectively. Thus, they propose three threshold values T_1, T_2, and T_3 to separate the pdf into four regions. For sessions with only one request, if the value of HS falls between threshold values T_1 and T_3, it is classified as a Web robot. Otherwise, the region that the session's HS value falls into is considered along with the additional metrics to classify a session. The second metric is *Diversity Factor* (DF), which captures how many diverse types of resources are requested per session. Rather than using the number of unique resources requested, the authors use unique resource sizes because modern Web pages deliver content to users through dynamic html, where the same html page

may deliver different embedded resources every time the page is requested. To define the metric, they let $R = \{r_i\}$ be the set of all resources requested in a given session. Then, the DF of a session is given by:

$$DF = \frac{|\{s|\exists s \in \mathbb{N}, s \in R\}|}{|\{r_i|r_i \neq 0\}|}$$

Through experimentation, the authors determine that sessions with $DF < 0.5$ are highly suspect to be from Web robots. They also propose an automatic way to find a threshold to classify sessions according to their DF measure, given by the K largest peaks in the pdf of DF across all sessions. The final metric is *Html Affinity* (HA), which reflects the assumption that human sessions will consist of a mix of resource types while Web robot sessions will be dominated by requests for content files that contain links to resources. HA is given by the percentage of requests in a session that are for an html, htm, jsp, or asp type resource.

These metrics are combined to develop an automatic classifier for Web robot traffic. First, the set of sessions are divided into those that have a single request and more than one request. Sessions whose HS do not fall between T_1 and T_3 are filtered out and considered to be P2SP traffic. For the remaining sessions, those whose DF is below the computed threshold value or whose HA is above 0.65 are considered to be Web robots.

They confirm that their scheme can detect all robots that request the file `robots.txt` from the logs of various Web servers hosted by Tsinghua University. They also confirm that the distribution of the metrics of the categorized traffic

are highly consistent across all sessions in a group, even across long time spans.

2.1.4 Analytical learning

Analytical techniques build on the traffic pattern analysis by developing formal models to represent the characteristics of web robot visits. A properly trained model ensures that a useful robot (i.e., one that does not convey random or arbitrary behavior) will be successfully detected. These analytical models are trained by using robot sessions extracted from the Web logs using a simple syntactic analysis or traffic pattern analysis technique. Analytical techniques can be further classified according to the modeling paradigm and the features of robot traffic considered in detection.

Detection using decision trees

Tan and Kumar attempt to discover Web robot sessions by utilizing a feature vector of the properties of Web sessions [112]. In the first step, they propose a new approach to extract sessions from log data. They argue that the standard approach based on grouping Web log entries according to their IP address and user-agent fields may not work well since an IP/user-agent pair may contain more than one session (for example, sessions created by Web users that share the same proxy server). Furthermore, a session containing multiple IP addresses or user-agents will become fragmented. Therefore, to determine what session a log entry

l belongs to, each active session is scanned to check the time difference between l and the current session, along with some unspecified session contiguity conditions. If this time difference exceeds a threshold or the conditions are not met, then a new session is generated starting with log entry l. Before scanning, all the active sessions are divided into four groups depending on whether the user-agent and IP address fields match those found in l. The first group consists of sessions where both user-agent and IP addresses match, followed by the two session groups with one matching field, and finally the group with no matching fields.

They then derive twenty-five different properties of each session by breaking down the sessions into episodes, where an episode corresponds to a request for an HTML file. Of these, they use only three for the initial classification of sessions. These include checking if *robots.txt* was accessed, the percentage of page requests made with the HEAD http method, and percentage of requests made with an unassigned referrer field. These attributes are used since they most distinctly represent sessions likely to be robots, assuming that normally a human user would not request *robots.txt*, send a large number of http HEAD requests, or send requests with unassigned referrer fields.

From this initial class labeling, the observed user-agent fields are partitioned into groups of known robots, known browsers, possible robots, and possible browsers in the following manner. If a derived session s contains a request for *robots.txt*, the session is declared to be a robot. Otherwise, the user-agent fields

of the requests in the session are considered. If s only ever has requests from one user agent, and the user agent is a known robot or a possible robot, then s is labeled as a robot. Otherwise, it is labeled as a human. If s has requests from multiple user agents, however, the session is labeled as a robot only if there are no sessions that are known browsers or possible browsers or if the session contains requests that all use the HEAD http method or requests that all have unassigned referrer fields.

Finally the technique adopts the C4.5 decision tree algorithm over the labeled human and robot sessions using all of the twenty-five derived navigational attributes. Their objective is to develop a good model to predict Web robot sessions based only on access features and to detect robot traffic as early as possible during a robot's visit to the site. This classification model when applied to a data set suggests that robots can be detected with more than 90% accuracy after four requests. They find that the recall (r) and precision (p) of their techniques after more than three requests are greater than 0.82 and 0.95, respectively.

Detection through neural networks

Bomhardt *et. al.* use a neural network to detect Web robots and compare the results to a decision tree technique similar to the one by Tan and Kumar [15]. They develop a Web log pre-processing tool called RDT in order to develop the feature vector for each session.

The RDT log pre-processing tool operates as follows: in the first step, the single log entries are broken into sessions by grouping requests that have matching IP addresses and user-agent fields such that the successive requests are less than 30 minutes apart. Next, the tool automatically classifies sessions using heuristics known to reliably identify Web robots. If a session contains a request with a login name, or the session IP address is contained in a pre-constructed list of known IP address for human users, the session is labeled as a human. If a session contains a request for a file from a given list of *trapfiles* that may be deployed (resources that should never be requested by a human, for example, typical files used in attacks such as `cmd.exe` or the file `robots.txt`), has a user-agent field that exists in a specified list of known robot agents, or if the session IP address is contained in a list of known robot IP addresses, the session is labeled as a robot.

Using this technique, the RDT is able to reliably generate a set of sessions known to be human or robot agents. Next, the attributes that constitute a feature vector are determined for each robot and human session. Specifically, they use the total number of page requests, percentage of image requests, percentage of html requets, total session time, average time between requests, average standard deviation of time between requests, percentage of error http responses, and percentage of http commands used to retrieve requests. They also include new features such as the total number of bytes sent, total site coverage during the session, and the percentage of responses sent with response codes 200, 2xx, 301,

302, and 304. Many of these attributes overlap with those used by Tan and Kumar [112], indicating an emerging consensus on what features should be used to identify Web robot traffic.

Machine learning models of human and Web robot activity are then built using a neural network, logistic regression, and a decision tree. The latter two were used to compare the performance of the neural network. Because several attributes in the feature vectors require at least two requests for their computation, the datasets were partitioned into three groups: *all sessions*, *single-request-sessions*, and sessions with more than two requests.

The authors evaluate their robot detection technique off of two log files, one from an educational website and another from an online shop. For each domain, the data was partitioned so that 40% is used to test the models during construction, 40% is used to train the models, and 20% is used to validate the models. They evaluate these models by measuring their recall, precision, and misclassification rates. The misclassification rates are also compared to a baseline misclassification rate achieved by a trivial model that always labels a session with the same class as sessions in the training data set. For the educational website dataset, they find that the neural network performs best when it is used only for sessions with more than two requests and a decision tree is used for sessions with a single request. The recall ($r = 0.942$) and precision ($p = 0.891$) achieved by this combined approach is comparable to the approach proposed by Tan and Kumar.

For the online shop dataset, the neural network offered the best performance after three or more page views were considered in a session, with $r = 0.947$, and $p = 0.954$.

Detection via a Bayesian network

Stassopoulou and Dikaiakos presented a Bayesian approach to crawler detection [108], in which they first identify sessions from the Web log by grouping entries according to their (IP, user-agent field) pairs. Requests are added to a session until the time between a request exceeds a certain timeout. The timeout threshold is dynamic, depending on the number of requests in the session. Then for each session the maximum sustained click-rate, session duration, percentage of image requests, percentage of pdf/ps requests, percentage of responses of 4xx code, and a binary value for if *robots.txt* were accessed are extracted. These features form the nodes (variables) of a naive Bayesian network, where the causal node C represents the session classification (human or robot) and each effected node F_i represents an extracted feature. This framework enables them to combine all pieces of evidence to derive a probability for each hypothesis (i.e., Web robot vs. human).

The network is trained over a data set of thousands of sessions. But rather than manually classifying each session as human or robot for training, they use a heuristic, semi-automatic method. First the sessions are assumed to be human.

Then the following heuristics are used to determine if a session must be labeled as a robot: (i) IP addresses of known robots; (ii) the presence of an http request for *robots.txt*; (iii) sessions that last longer than three hours; and (iv) an HTML-to-image request ratio of more than 10 HTML files per image file. These metrics were then extracted for the heuristically classified sessions and those modeled by continuous distributions such as session duration were quantized into discrete values using the entropy method [104]. The initial values for the root node as well as the conditional probability distributions for the non-root nodes were computed from the extracted traffic features. The experiment was performed five separate times, with each trial using different levels of human and robot sessions in the training sets. Their results show a best recall $r = 0.95$ (when the training set contains an equal number of humans and robots) and at its worst a recall of $r = 0.80$ (when the number of human sessions in the training set dominate the number of robot sessions).

Detection by a Hidden Markov model

Lu and Yu use a hidden Markov model (HMM) to distinguish robot and human users based on request arrival patterns [82]. They argue that a human visitor to a Web page is characterized by a burst of http requests from the browser for all embedded resources, followed by a period of inactivity while the user views the page. A robot, however, sends requests for resources at a slower rate and

with a steady period between requests. To capture this phenomenon, the authors partition time into discrete intervals of the same length. One or more requests from the same user that arrive in the same time interval is called a batch arrival. Such a batch arrival is more reflective of a human user than a robot. Each interval with a batch arrival is considered as an observation for the HMM.

The authors use previously observed robot sequences to train their HMM. This training set was obtained by examining the user-agent field and extracting those requests from agents that are clearly robots. Only batch arrivals with more than 30 requests were considered for training, resulting in 612 observed sequences. Future incoming request sequences are fed as input to the trained HMM to compute the likelihood that the request sequence is from a robot. This likelihood is computed by the forward-backward procedure [98]. To test the adequacy of their technique, an equal mix of robot and human sessions from the same logs as the training set were composed to form the test data. For this test data, their HMM yields a detection rate of 0.976 with only 0.02 false positive rate.

2.1.5 Turing test systems

Turing test systems attempt to classify active sessions in real time by forcing the visitor to take a test in order to determine if the session is human or robot induced, rather than doing a post-mortem analysis of server access logs. The key aspect of these systems is the technique employed to force a user into taking the Turing

test.

Detection via CAPTCHA tests

A common example of a Turing test system is the CAPTCHA test, proposed by Ahn *et. al.* [3]. CAPTCHA, which stands for **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part, is a challenge response test embedded in an html page. The server generates a simple test which the user must pass in order to gain access to some resource. Common CAPTCHA tests charge users to copy text from a generated image or type a word or a phrase from a generated sound file. These images and sound files are generated so that computers are unable to recognize the characters from the image or analyze the sound file to determine the word or the phrase spoken. CAPTCHA tests have been employed with great success to prevent robots from gaining access to Web forums, where a robot can easily harvest e-mail and IP addresses of users that post to the forums as well as spam unwanted advertisements on discussion threads. CAPTCHA is also commonly used on pages to create accounts for e-business sites and signing up for online e-mail services.

It is not impossible for a human user to fail a CAPTCHA or to request the server to generate a new test, if the image or the audio distortion is too great to interpret. A human user may also get discouraged if the test is too difficult, and may even terminate the session. Furthermore, it is possible that humans

who are only interested in browsing the site for resources accessible without the CAPTCHA test may not take it. Thus, while CAPTCHA can provably classify active sessions that are produced by humans, failing or not taking a CAPTCHA test is not sufficient evidence to classify a session as robot induced.

Building stronger and more sophisticated CAPTCHA systems that are easier for humans to pass, more difficult for Web robots to break through, and applicable in a wider range of domains is an area of active research. This is due to the existence of CAPTCHA-solving services, which Web robots could query in order to bypass a test [90]. Shirali-Shahreza *et. al.* introduce a CAPTCHA designed to counter SMS-Spam messages sent to cell phones [106]. When an SMS message is sent, the carrier's SMS routing agent sends a challenge to the sender in the form of an image. The SMS sender then needs to reply with the correct name of the object in the picture before the routing agent forwards the message to the recipient. Gossweiler *et. al.* introduced a new CAPTCHA test that charges users to re-orient an image such that it is facing upright [48]. This task requires a visual analysis of the patterns in the image in order to perform a geometric modification. Image reorientation can be easily performed by humans compared to CAPTCHAs that use distorted sounds. Such reorientation, however, cannot be performed easily by automated agents. Kluever *et. al.* present a CAPTCHA that challenges the user to label a video and then compare the user input to the video's tags from YouTube [73]. Responses are graded based on the string edit distance from the

tags available, and through stemming the user responses to improve the likelihood that responses match YouTube tags. They find that the success and failure rate of their video-based CAPTCHA is similar to the traditional CAPTCHA which involves transcribing distorted text, but a majority of participants find the video CAPTCHA to be more enjoyable.

Detection by implicit human browsing behavior

While a Web robot is capable of exhibiting human-like patterns in its traffic, it can still be captured by a turing test system that checks for behavior unique to an interface that humans use to access information on the Web. Park *et. al.* exploit this by developing a technique which requires users to take a Turing test implicitly during a session. In this case, human activity is detected by noting the occurrence of specific events caused by a Web browser [95]. Their technique is also able to differentiate human traffic from robots that request html pages and their embedded resources.

To detect events caused by human activity, for each client request a random key k is generated and paired with the resource requested. This pair is saved in a table indexed by the IP address that the request comes from. A custom Javascript is embedded into the pages served to the client dynamically. The Javascript includes an event handler for mouse movement or key clicks. This event handler will fetch a fake embedded object whose URL contains k. When

the event handler sends the request for the fake embedded object, if the k in the URL matches the value of k stored in the server-side table, the session is classified as human. If k does not match, or the fake embedded object is never requested, the session is classified as a robot. The script is obfuscated with additional entries to prevent it from being deciphered by a robot attempting to discover the proper value of k. This test determines if a user is human or robot by determining if mouse or key clicks are ever performed on the Web site.

In practice, some users may disable JavaScript on their browsers. To avoid classifying such users as robots, common patterns of Web browsers are considered as well in a second embedded Turing test. The second technique is based on the observation that many robots crawling specific types of resources, such as only html or only image files, do not download certain objects on a Web page, particularly resources containing html presentation information such as CSS or JavaScript files. To administer this test, a reference to a non-existent CSS file for each html page is dynamically added upon being served. While Web browsers should send a request for this CSS file automatically, robots that do not download presentation related information will not send a request and thus be classified. Similarly, links to 1-pixel transparent images and silent audio files are also placed as transparent resources in the html. Because the link is invisible humans should not request these resources, while robots may analyze the html and make requests.

This technique is implemented in the CoDeeN content distribution net-

work [95]. The results show that 95% of human users are detected within the first 57 requests made, with a false positive rate of 2.4%. These results confirm that using both techniques together accurately separates robots requesting html and embedded resources from human visitors. Robots requesting non-html documents exclusively during a session are not detected, however, as only clients that request html documents become subject to the implicit Turing test.

Detection of DDoS botnet attacks

Kandula *et. al.* presented a system implemented as an extension to the Linux kernel, called Kill-Bots, that is designed to protect Web servers from DDoS attacks posing as a flash crowd event [69]. A flash crowd event is a phenomenon where the server experiences a sudden surge in the number of requests it receives. Such DDoS attacks may be launched by *zombies*, a type of Web robot installed on infected computers and controlled by a single mastermind. This mastermind usually has direct control of thousands or hundreds of thousands of zombie robots which are all directed to execute the same command. In this case, the zombie robots to be detected are those used to execute a DDoS attack on a Web server.

Kill-bots is a two function system, combining user authentication with admission control. The two-stage authentication mechanism is activated when the server becomes overloaded with active sessions. First, a CAPTCHA test is administered before any site resources can be accessed. While legitimate users will solve

the test or at least try to solve the test several times before getting frustrated and leave the site, zombies will continue to send new requests without attempting to solve the test. As a result, if the number of unsolved puzzles from a session exceeds a certain threshold, the IP address of the session is logged as a zombie and future requests from that IP address are discarded. The second stage of authentication is reached when the size of the set of zombie IP addresses stabilizes. In this stage, the CAPTCHA test is no longer administered and the Bloom filter is relied on exclusively to determine if incoming requests from an IP address should be dropped. This stage enables legitimate users who failed the CAPTCHA test and left in discouragement to return and use the site even during a DDoS attack.

If too many server resources are consumed by the authentication process, the remaining available resources may not be sufficient to provide a reasonable quality of service to the human users who are successfully authenticated into the system. To address this problem, the second function of Kill-bots is an admission control system. The admission control will serve puzzles to new, unauthenticated users with an admission probability α^*, calculated as a function of the current arrival rate of attacking requests, legitimate requests, average time to serve puzzles and requests, average number of requests per legitimate session, fraction of the time the server spends in authentication and serving, and the idle time. A request from an IP address which has not yet been authenticated with the server but not blocked will be granted access with probability α^* and dropped with probability

$1 - \alpha^*$. Being adaptive, α^* self adjusts as the values of the parameters change during an attack so that server goodput is always maximized.

The system was evaluated by launching a botnet attack on a local Web server using the PlanetLab wide area network [117]. Legitimate clients were also setup on the same LAN as the Web server. The results show that their system is able to quickly adapt during the first stage of the authentication process, causing the mean response time of requests to expeditiously drop to a value close to the one when there is no attack. When the second part of the authentication process is activated, a large increase in goodput is observed due to reduced detection cost and users accessing the system that previously could not pass the CAPTCHA test. Despite the ongoing DDoS attack, the server performance is close to the performance when it is not under attack. Thus, the Kill-Bot system can properly drop all requests from zombies and produce a correct list of all sessions induced by them.

2.1.6 Limitations in the state-of-the-art

The techniques presented in this survey reflect major milestones in Web robot detection over the past decade and also demonstrate how widely varied detection methods can lead to an effective solution. These techniques have evolved from simple access log parsing, to using analytical models, to solutions engineered for specific types of robots and traffic. In this section, we first compare the exist-

ing classes of techniques to identify which philosophy may be most effective at detecting both common and unknown, evasive robots. Subsequently, we identify the issues involved in the implementation of this detection philosophy that we conjecture to be the most effective.

It is difficult for a third party to experimentally compare the results of these published detection techniques for two reasons. First, each detection technique relies on specialized tools or operating system modifications, which would need to be implemented faithfully. For sophisticated system-level modifications, such as those used in Kill-bots, a third party implementation cannot be guaranteed to have high fidelity. Second, it is expected that the accuracy of some techniques will depend on the data set used for demonstration. For example, the technique based on navigational patterns may exhibit poor accuracy if the data is from a Web site with very few pages, making the navigational pattern of a robot similar to most humans because they are likely to cover the majority of a site during a session. Furthermore, because the sessions are so short on these types of Web sites, there may not be enough observations for learning techniques to reach valid conclusions. Thus, at least a syntactic analysis approach may be the most effective type on such Web sites. As another example, consider an access log from a very large Web site that contains entries from a variety of robots with different navigational patterns. In this case, the navigational pattern and learning techniques will naturally be more effective than syntactic log analysis. These examples illustrate that the

prevalent techniques will need to be compared using a variety of logs with varying features and levels of robot activity to obtain any meaningful conclusions. These data-intensive demands, the need for proper operating environments and hardware for each technique, and the lack of implementation details to faithfully reproduce each detection strategy make a third party comparison of the existing techniques unrealistic.

The works presented can thus be best compared by considering the strengths and weaknesses of their underlying detection philosophy. Syntactic log analysis relies only on a textual analysis of the Web logs that considers only explicit recorded information. Traffic analysis techniques improve upon syntactic analysis by considering important facets about a robot's traffic that may be implicit in the logs. Analytical learning models are considered to be even stronger than traffic analysis as they are able to consider the relationships among these implicit facets that are considered separately in traffic pattern analysis. Turing test systems may be considered to be strongest, as they may be able to classify all robots of a specific type on the Web server in real time. Because they are engineered to detect only specific types of robots (such as those used in DDoS attacks or those that request at least one html document), however, their power is limited.

Analytical learning techniques may also be superior in the detection of evasive robots compared to the other three types of techniques (syntactic log analysis, traffic pattern analysis and Turing test systems) This is because the other three

techniques rely on some procedure, algorithm, or system which a savvy author can engineer a robot to evade. The analytical detection techniques, however, do not incur this disadvantage because they rely on learning the features of robot traffic on the server. As a result, it may be difficult for robot authors to circumvent detection by the analytical techniques. In fact, the only way to avoid detection by an analytical learning technique is to have a robot emulate the visiting patterns of a human, something that is not simple to achieve. For example, human users frequently traverse back and forth between pages, reference information, and/or navigate back to a main page to click on a different link. Such back-and-forth navigation cannot be accurately emulated by robots, unless a robot was programmed to follow a specific sequence of requests and wait a specific duration between requests that mimic a human visitor. Furthermore, a robot that crawls like a human would discover only a small amount of information, which may or may not be meaningful. Moreover, robots that mimic humans will not impose any undue strain on the Web server. Hence, we should be comfortable allowing such robot traffic to go undetected. In summary, our discussion suggests that analytical learning models are the most promising types of robot detection techniques because they not only consider implicit characteristics of robot behavior but also try to explore the relationship between these implicit characteristics using formal models.

2.2 An Integrated Approach for Offline and Real-Time Detection

Our above analysis finds analytical models to be the most promising approach for offline Web robot detection. However, although offline and online approaches are synergistic and offer complementary benefits, they have been developed independently, with disparate underlying philosophies. Real-time approaches also require extensive modifications to Web servers, limiting their practical use, and lack the ability to identify more general classes of ill-behaving robots. Furthermore, recent evidence suggests that Web robot traffic is in a constant state of evolution [31]. This evolution is natural considering how rapidly new value-added and social services are being developed, which require support from more advanced Web robots to fetch information. This has forced Web robots to make more frequent, demanding crawls with sophisticated techniques in order to stay abreast with the latest available information. Thus, an analytical detection scheme that relies on a snapshot of robot traffic at any given point of time is sure to become less effective over time. To reliably detect Web robot sessions despite their constant state evolution, we need a new analytic approach, capable of operating both offline and in real-time, that is based on *fundamental distinctions* between Web robot and human traffic.

In this section, we present a novel, integrated approach to detect Web robots. The approach relies on training analytical models over the request patterns of robots and humans for different types of resources. We motivate why the resource

request patterns of robots and humans are intrinsic to their respective behaviors, and hence, are fundamentally distinct from each other. We also describe how the contrasts in these request patterns are likely to be immutable and persistent, even if robots and humans change their behaviors over time. We evaluate the performance of our methodology for both offline and real-time detection by playing back multiple streams of robot and human sessions collected from the University of Connecticut (UConn) School of Engineering (SoE) Web server. The results indicate that our approach can separate robots from humans with high accuracy for varying session lengths and proportions of robot traffic. Our approach thus offers two key advantages over contemporary detection techniques, namely, it: (i) defines a model based on a fundamental, time-invariant difference between robot and human behavior; and (ii) can be simultaneously used for highly accurate offline and real-time detection.

2.2.1 Resource request patterns

We hypothesize that the patterns of requests for resources made within robot and human traffic are distinct, and is not expected to change over time desptie the constant evolution of Web robots. This distinction arises because human users retrieve information off of the Web via some interface, such as a Web browser. This interface forces the user's session to request additional resources automatically. Most Web browsers, for example, retrieve the html page requested, parse through

it, and then send a barrage of requests to the server for embedded resources on the page such as images, streaming videos, and client side scripts to execute. These scripts may also cause additional artifacts on the page to be requested and then displayed to the user, such as pop-up windows or advertisements within the Web page being viewed. Thus, the temporal resource request patterns of human visitors are best represented as short bursts of a large volume of requests followed by a period of little activity. In contrast, Web robots are able to make their own decisions about what resources linked on an html to request and may choose to execute the scripts available on a site if they have the capacity to do so.

Defining Request Patterns

We define a session as a sequence of requests from the same IP address and user-agent field, where the time between any two consecutive requests is less than 30 minutes [86]. Subsequently, we define a request pattern as an ordered sequence of specific resources that are requested in a session. Because a Web server hosts numerous unique resources, a pattern that lists specific resources will be distinct and unique for each session. Therefore, this representation will neither identify the similarities across robot (human) sessions nor expose the differences between robot and human sessions. Therefore, to expose these similarities and differences, we generalize the notion of a resource request pattern, and represent it in terms of resource *types* rather than specific resources.

For this generalized representation, we classified the resources that Web servers may host into a small number of distinct categories. We collected Web server access logs from the UConn SoE Web Server that captured all requests between the years 2007 and 2009. Through a comprehensive manual analysis of this log, we identified 172 unique resource extensions. We conjecture that this two-year log captures most if not all the resources hosted on the UConn server because the: (i) length of the log is extensive; (ii) number of unique resource extensions discovered is low; and (iii) regular updates to the SoE web server only consist of news articles and changes to faculty sites, which do not introduce any new resource extensions.

We partitioned these extensions into nine classes, summarized in Table 2.2. The table also shows representative resource extensions assigned to each class. For example, txt, css, and xml files are assigned to the *txt* class. html, php, jsp, and cgi files, used by a Web browser to display information and execute scripts, are assigned to the *web* class. Extensions classified into the *doc* class include rich-text documents that contain information that is formatted to be read by specific software.

To assess whether our scheme reasonably classifies the resources on the SoE server, we examined whether the distribution of requests for the various types of resources is consistent with how resources might be requested from the UConn Web server. This distribution, shown in Figure 2.2, indicates that over half of

Class	Extensions
text	txt, xml, sty, tex, cpp, java
web	html,htm, asp, jsp, php, cgi, js
img	png, tiff, jpg, ico, raw, pgm
doc	xls, doc, ppt, pdf, ps, dvi
av	avi, mp3, wmv, mpg, wmv
prog	exe, dll, dat, msi, jar
compressed	zip, rar, gzip, tar, gz, 7z
malformed	request strings that are not well-formed
noExtension	request for directory contents

Table 2.2: Example class-wise assignment of resources

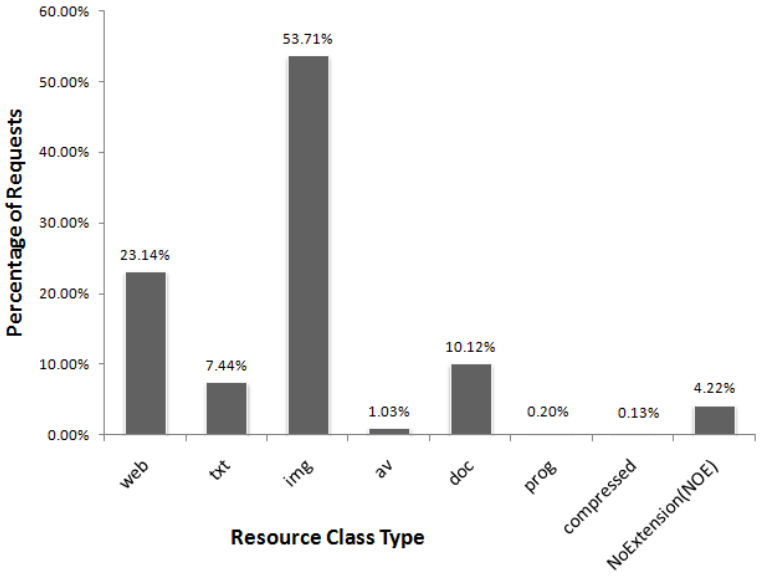


Fig. 2.2: Distribution of resource requests for SoE server

the requests are for *img* resources, almost one quarter are for *web* resources, and nearly one-fifth are for *txt* and *doc* resources taken together. This distribution is consistent with what might be expected from the SoE server because it primarily hosts two types of pages. The first type includes pages that convey information about the school, and published news articles about faculty, students, and departments. These commonly accessed pages primarily feature *web* and *img* resource types. The second type include faculty, lab, and student pages that share information about faculty members and research labs, and disseminate publications and data. Such pages include a combination of *web, img*, and *doc* resources. Finally, the percentage of *noe* requests may be accounted for by external links that point to root folders which redirect to an html page. Thus, these nine classes can accurately explain the frequencies with which different types of resources may be requested from the SoE server. We note that these nine classes may not always be adequate to group resources hosted on Web servers from all domains. However, we believe that they can provide a solid starting point for customizing a domain-specific resource classification scheme.

2.2.2 Distinguishing robots from humans

We argue that the resource request patterns of Web robots will be inherently distinct from those of humans. Moreover, these intrinsic contrasts between the request patterns of robots and humans are likely to be immutable despite any

evolution in their behaviors. This is because these contrasts are grounded in the different ways in which robots and humans interact with Web sites. Humans interact through Web browsers that retrieve and parse the requested html pages and then send a barrage of requests for resources embedded on those pages. These embedded resources include images, streaming videos, and client-side scripts. Because human interactions are driven by browsers, we expect little variation in the request patterns from one user to another and from one session to the next. Furthermore, even if the behavior of these browsers were to change, request patterns would still be limited by the structure of the Web site. For example, a human visitor will generally not request a *doc* file, before requesting either the *web* file which links the *doc* file or another resource that is also embedded on the same Web page. On the contrary, all robots regardless of their functionality, workload, or crawling algorithm, make individual decisions about what resources to request [33]. Thus, they may only request a single type of resource during a session, may not request the resources embedded within an html page, and depending on their offline knowledge of a Web site, may request two consecutive resources located on different pages. Thus, the resource request pattern of Web robots will show great variability and flexibility while those of humans will be rigid because they are constrained by the well-understood behavior of Web browsers and structures of Web sites.

Table 2.3 shows sample sequences of resource requests from different ses-

sions. These sequences illustrate the differences between request patterns of humans and robots. A typical human session starts with a request for either a *web* or a *noe* resource and is followed by a number of requests for *txt* and *img* files. Following a request for a *web* or a *noe* resource, a human session sends a number of requests for other *web*, *txt*, and *img* resources. These *txt* resources may correspond to css, and xml files that are used by the browser to define the look and feel of the page, while the *img* requests may correspond to the embedded images within the page. In contrast, a robot session starts with a request for a *web*, a *txt*, or a *noe* resource, and the subsequent order of resource requests follows no particular pattern. For example, the request pattern of MJI2bot shows that it almost exclusively requests *web* resources, suggesting that it operates as an indexer robot for search engines. ICC-Crawler, on the other hand, exhibits a different pattern where it sends a number of requests for *noe* resources. Such behavior could mean that a robot is sending malformed requests or is trying to access directory listings at the site. In summary, we see that the resource request patterns of humans are similar across sessions, whereas the request patterns of robots vary between robots and are also different from human sessions.

We also expect this contrast in the resource request patterns between robots and humans to persist unless Web robots voluntarily restrict their behavior to appear like humans. Robots that behave like humans, however, are unlikely because it requires them to request numerous useless html and embedded resources. Fur-

User	Sample Request Pattern
Human	noe,img,img,img,img,img,img,img,img,img,img,img,img,img,img
Human	web,img,img,img,web,img,img,img,web,img,img,img,web,img
Human	noe,txt,txt,img,img,txt,img,img,img,img,img,img,img,img,img,img
Human	noe,txt,txt,img,img,txt,img,img,img,img,img,img,img,img,img,img,img
MJ12bot	web,web,noe,web,web,txt,web,web,web,web,web,web,web,web,web
Yandex	txt,noe,noe,txt,doc,doc,web,doc,web,doc,doc,web,web,web,web
W3C-checklink	txt,noe,txt,txt,noe,web,txt,noe,txt,noe,txt,noe,txt,txt,noe
ICC-Crawler	noe,web,web,web,noe,noe,noe,noe,noe,noe,noe,noe,noe,web,web

Table 2.3: Sample resource request patterns of robots and humans

thermore, human-like robots must adapt to short sessions and limit the intensity of their request rates to mimic the inter-request think times of humans [83]. Modern robots that seek the dynamic and time-sensitive information that users share on Web sites and social media services would find such behaviors inefficient. In summary, because of these fundamental and immutable differences in the resource request patterns between robots and humans, a detection approach rooted in identifying these differences may be accurate, reliable, and also stand the test of time.

2.2.3 Detection approach

In this section, we introduce a discrete time Markov chain (DTMC) model [115] to represent resource request patterns. Subsequently, we discuss how the DTMC model can be used to detect Web robots.

Model Representation

We identify a resource request pattern for each session based on the resource classification scheme defined in Section 2.2.2. Diagrammatically, we then represent all possible resource request patterns in a session using a *resource request graph*, shown in Figure 2.3. A session begins at one of the nodes of the graph and at each time step either moves to another node or stays at the current node. The progression of a collection of sessions along this graph can be modeled using a DTMC where the states represent requests for specific types of resources and the transitions represent the probability that a request for one type is followed by a request for another type.

Mathematically, the DTMC may be represented as a tuple $(\mathbf{s}, \mathbf{P})$ where the i^{th} element of the entrance vector $\mathbf{s}$ represents the probability that a session starts at resource type i, and $p_{i,j}$, which is the $(i,j)^{th}$ element of the transition matrix $\mathbf{P}$ is the probability that a request for resource type j follows a request for resource type i. Transition probabilities can be estimated as $p_{i,j} = r_{i,j}/m_i$, where $r_{i,j}$ is the number of times resource type j is requested after type i, and m_i is the number of times resource type i is requested across all sessions. Similarly, $\mathbf{s}_i$ is estimated as c_i/n, where c_i is the number of sessions that start with a request for resource type i and n is the total number of sessions considered. $r_{i,j}$, m_i, and c_i can be estimated using the data contained within the Web server access log. Through these definitions, $(\mathbf{s}, \mathbf{P})$ characterizes the resource request pattern of the sessions

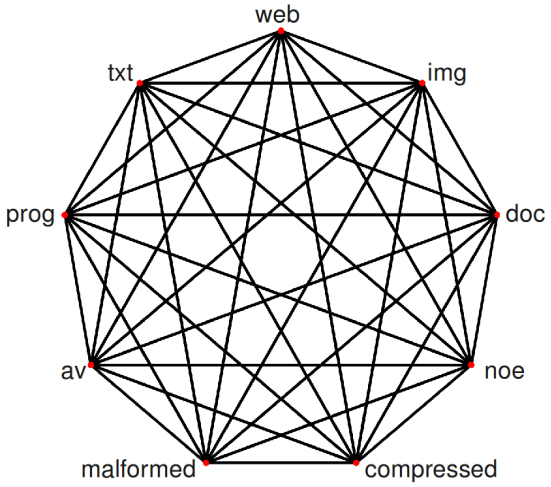


Fig. 2.3: Resource request graph

used to estimate **s** and **P**.

The Markov assumption underlying the DTMC model may be rational because a dependence between the requests for *types* of resources is unlikely for both robots and humans. Human requests for resource types depend at most on the previous request. For example, human sessions session will not request a *doc* resource on a Web page, before requesting the *web* resource that links the document. Similarly, typical Web robots that crawl using depth- and breadth-first algorithms [85] decide which resource to request after considering only the previous one.

Detection Algorithm

To detect whether a given session arises from a robot or a human, we define $\mathcal{S} = (x^1, x^2, ..., x^n)$ as the resource request pattern of a session with n requests.

The log-probability that the DTMC $(\mathbf{s}, \mathbf{P})$ will generate $\mathcal{S}$ is given by:

$$\log Pr(\mathcal{S}|\mathbf{s}, \mathbf{P}) = \log \mathbf{s}_{x^1} + \sum_{i=2}^{n} \log p_{x^{i-1},x^i}$$

Let $\mathbb{R} = (\mathbf{s}_r, \mathbf{P}_r)$ denote the DTMC that represents Web robot sessions and $\mathbb{H} = (\mathbf{s}_h, \mathbf{P}_h)$ denote the DTMC that represents human sessions. Since $\log a > \log b$ iff $a > b$, if $\log Pr(\mathcal{S}|\mathbb{R}) > \log Pr(\mathcal{S}|\mathbb{H})$, $\mathcal{S}$ is more likely to have been generated by $\mathbb{R}$, and should be classified as being generated by a robot. Otherwise, $\mathcal{S}$ should be classified as a human.

The approach aggregates log-probabilities as a sum instead of collecting raw probabilities as a product in order to avoid multiplying many small values, which may lead to an underflow in computation. Moreover, these log-probabilities can be used to differentiate between robot and human sessions because such differentiation only requires us to consider the relative difference in the probabilities that the session may be generated by $\mathbb{H}$ or $\mathbb{R}$, and not their actual values.

Model Evaluation

We evaluate the performance of our detector using the following, standard set of metrics that are commonly used to evaluate the performance of a binary classifier [113]:

- **Recall (r)**: Recall is calculated as the the total number of robot sessions identified by the detector divided by the number of true positive (robot sessions correctly classified as robot) and false negative (robot sessions wrongly

classified as human) classifications. It thus represents the fraction of all robots in a test set that were correctly identified, and captures the ability of the detector to positively identify robot sessions. The false negative rate of the detector is thus given by $1 - r$.

- **Precision (p)**: Precision is computed as the actual total number of robot sessions identified by the detector divided by the number of true positive and false positive (human sessions wrongly classified as a robot) classifications. It captures the detector's propensity to incorrectly label a human session as a robot. The false positive rate of the detector is thus given by $1 - p$.
- **F_1 measure:** The F_1 score is a measure of the detector's overall accuracy which combines both its precision and recall into a single value [101]. It is defined as the harmonic mean of precision and recall, so that F_1 approaches its highest value of 1 if and only if both the false positive and false negative rates of the detector approach 0. On the other hand, F_1 approaches its lowest value of 0 as *either* the false positive or false negative rate approach 1. Because our detector must have low false positive and negative rates, we use the F_1 measure to evaluate its overall quality in addition to precision and recall.

Model Training

We generated a data set from the access logs from the UConn SoE Web server from the period April 2008 to August 2008 to train the model. Typically, data sets used to train analytical robot detection models are generated by manually labeling sessions as robot or human [107]. In this manual approach, an expert examines the IP address or user-agent field of a request, and uses prior knowledge and practical experience to identify those values that correspond to robots. While this manual examination can distinguish between robot and human sessions with high accuracy; the process is cumbersome, time-consuming and limits the number of sessions that can be labeled. Moreover, the labeling is limited only to the knowledge of a few experts. Therefore, to generate a larger training set, we developed an automated, heuristic procedure, which relies on a database of regular expressions that represent user-agent fields of Web robots. By using this database, taken from version 6.7 of the tool *AWStats* [9], we expect to produce a training set that is more inclusive because it incorporates the collective knowledge of many administrators, researchers, and other experts that have contributed to building this database. The automated procedure comprises the following steps shown in Figure 2.4:

1. If the user-agent field of a session matches an entry in the database, label the session as robot.
2. If the IP address of a session matches an IP address within the UConn

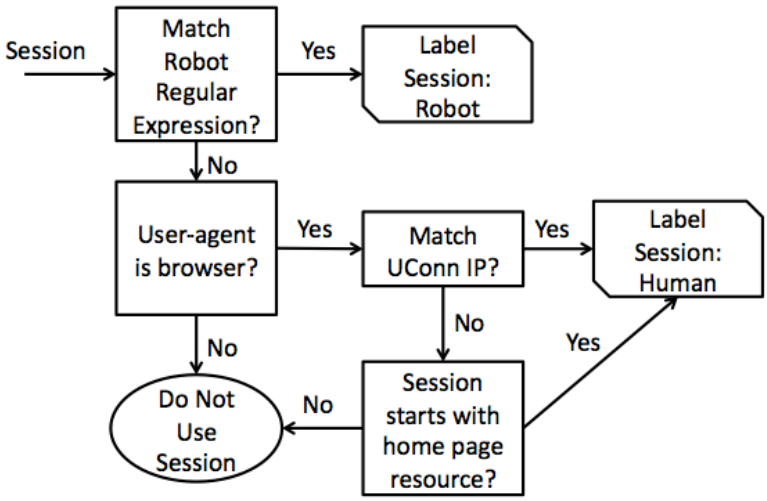
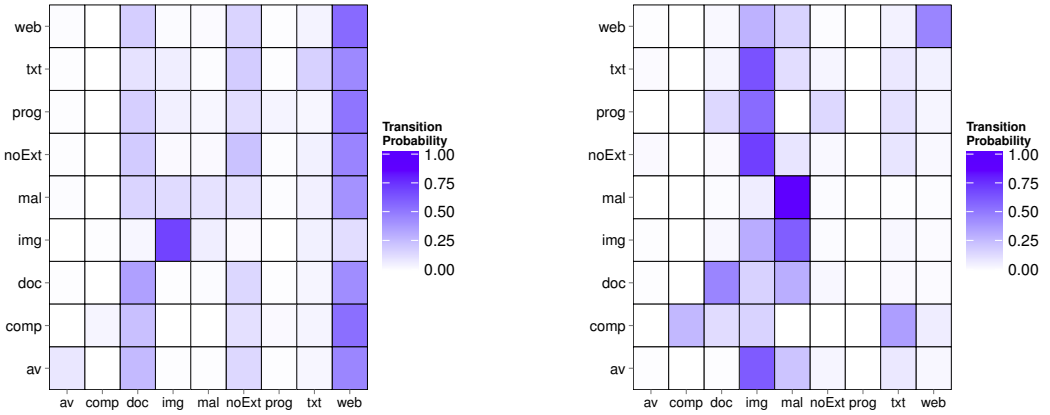


Fig. 2.4: Heuristic procedure to build training sets

domain, and its user-agent field matches one used by a common browser, label the session as human.

3. If the IP address of a session is outside of UConn, the user-agent field matches one used by a common browser, the initial request of the session is for the home page, and a subsequent request follows within 1 second for an embedded resource on the home page, label the session as human.
4. If none of the above rules apply, do not include the session in the training set.

Step 3 comprises a verification sequence to ensure that the human sessions added to the training set are genuine. Under this verification sequence, a session is admitted as human only if the user user-agent field of a request matches that of a Web browser, the first request is for the homepage, *and* a subsequent request



(a) Robot DTMC Transitions

(b) Human DTMC Transitions

Fig. 2.5: Trained detection models

for an embedded resource on the home page arrives within 1 second. We impose the 1 second threshold to ensure that the resources embedded within the page are requested immediately following the home page, which is a signature behavior of Web browsers. Thus, to apply this verification sequence, we compiled a list of embedded resources on the site's homepage. We used this heuristic procedure to label sessions that were randomly sampled from the Web access log. We repeated this random sampling until we collected 1000 robot and 1000 human sessions for the training set. We chose to collect an equal number of human and robot sessions because a previous study [108] suggested that an analytical model trained with an equal number of robot and human sessions provides the highest level of detection accuracy.

To assess whether the heuristic procedure differentiates between robot and human sessions accurately, we examine the human and robot DTMCs trained using this set. We reason that if the sessions are labeled accurately then the transition probabilities of these DTMCs should embody distinct behaviors of these two types of visitors. Figure 2.5(a) shows that the robot DTMC features a high probability of consecutive requests to *img* files and subsequent requests to a *web* resource following a request for any type of file. Multiple *img* file requests likely correspond to indexer robots who only visit the site to collect images for search engines. We find that *web* files are likely to be requested following any other type of resource request, which matches the behavior of robots who request a *web* resource to identify new links after collecting all files of interest on the present page. We also observe a weaker, but still non-negligible probability of subsequent requests to *doc*, *noExtension*, and *txt* resources, irrespective of the type of the previously requested resource. Since these are the most common types of resources hosted on the Web site as seen in Figure 2.2, this pattern matches the behavior of more general indexer robots who send requests for all kinds of available files.

The human DTMC in Figure 2.5(b) shows a decidedly different pattern of behavior compared to Web robots. Here, we find a high probability that *web*, *img*, $malformed$, and *txt* resources are requested following a *web* resource. This pattern correlates with the expected behavior of Web browsers that send requests for embedded resources after an html page is retrieved. We also see that *img* files

are requested after *txt*, *prog*, *noExtension*, other *img* files, or *av* files. In addition, unlike the robot DTMC, the human DTMC contains near zero probabilities for submitting a *noExtension* request. This is because although robots may wish to list the directory of a Web page by sending a *noExtension* request, humans almost always submit requests for specific resources instead of requesting a directory listing.

2.2.4 Offline detection analysis

In this section, we summarize the performance of the DTMC models when used to detect robots offline. In offline detection, we tested the detector with seven different data sets, each extracted from a month-long access log. Each test data set consisted of $10,000$ sessions divided equally among Web robots and humans. In offline detection, the detector analyzed all the resource requests in a session before classifying it as robot or human.

For all the seven test sets, we find that both precision and recall lie approximately between 0.8 and 0.86. In other words, the detector can correctly identify between 80 and 86% of all the robots. The F_1 measure lies between 0.828 and 0.853 for every test set, confirming that both recall and precision are high and that the detector has overall excellent performance.

In these preliminary experiments, the data sets are equally balanced between robot and human sessions to illustrate the potential of the detector. In practice,

Test Set	r	p	F_1
Feb. 2008	0.8072	0.8502	0.8282
Mar. 2008	0.8374	0.8531	0.8452
Oct. 2008	0.8418	0.8602	0.8509
Nov. 2008	0.8436	0.8620	0.8527
Dec. 2008	0.8498	0.8429	0.8463

Table 2.4: Performance metrics for offline detection

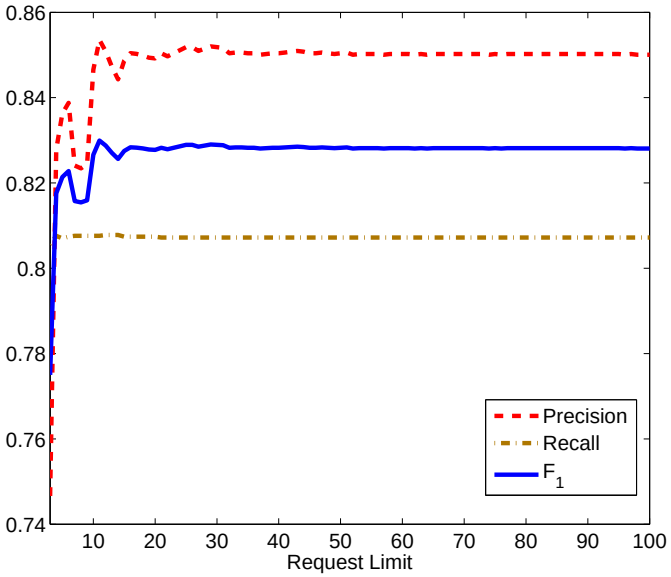


Fig. 2.6: Evaluation metrics for varying session length

however, the detector must identify robots from different profiles of Web traffic, which may vary along at least two dimensions: (i) number of requests in each session (session length) and (ii) proportions of robot and human sessions (session mixture). Therefore, we assess the performance of the detector by varying these two properties of the test data sets.

Varying Session Length

In the first set of experiments, we expected the detector to classify a session after analyzing a pre-defined number of requests, without waiting for the session to terminate. Figure 2.6 shows the performance metrics for the February 2008 test set. The metrics for the other sets are not shown because they exhibited similar trends. The recall metric remains stable across the different session lengths because the log probabilities $Pr(\mathcal{S}|\mathbb{R})$ and $Pr(\mathcal{S}|\mathbb{H})$ quickly converge. Precision peaks at 0.855 after 12 requests and declines steadily until about 20 requests after which it stabilizes. This suggests, perhaps contrary to the common belief, that analyzing additional requests within a session does not improve the false positive rate.

These experiments show how the detector can reliably identify most robot sessions within just a few requests. Furthermore, it takes only 12 requests for the F_1 metric to reach within 0.1 of its steady-state value. In other words, the detector is capable of distinguishing between robot and human sessions so long as each session contains at least 12 requests, which is likely for sessions on most Web domains. In human sessions, Web browsers will request every resource embedded on a page, and the number of such resources on a single page is typically much greater than 12[2]. Similarly, an average robot session contains at least hundreds of requests [4]. Thus, the detector is expected to accurately classify sessions despite

[2] `https://developers.google.com/speed/articles/web-metrics`

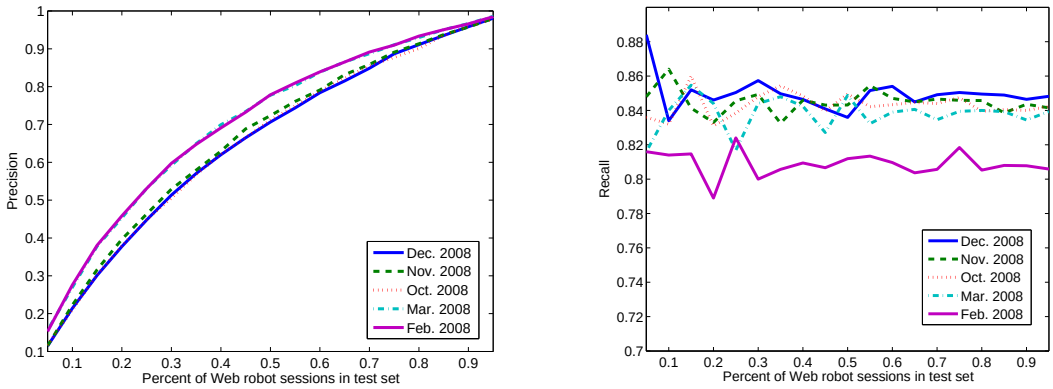
their varying lengths.

Varying Session Mixture

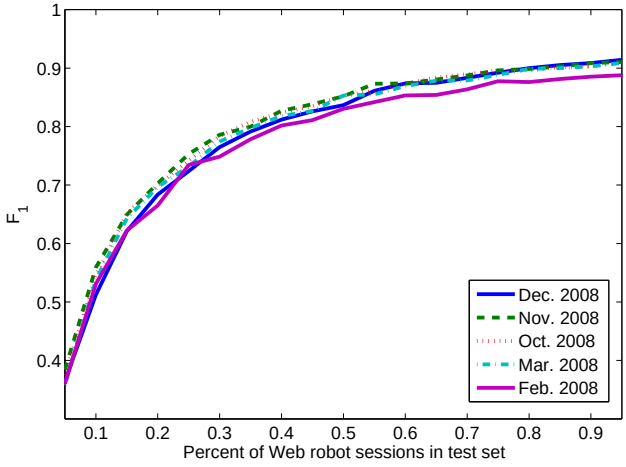
Next, we evaluated the performance of the detector over test sets with varying proportions of robot and human sessions. For every month-long test set, we changed the proportion of robot and human sessions by randomly deleting or duplicating robot sessions until a desired mixture was achieved. Thus, for each month, we created an additional 9 data sets, with the proportion of robot sessions varying between 10% and 90%, in steps of 10%. We assessed the performance of the detector for all these data sets.

Unlike the session length experiments, performance was slightly different for each month, so we report the results for all the 9 data sets generated for each month in Figure 2.7. In Figure 2.7(c), we see how the F_1 measure rapidly improves as the proportion of robot traffic increases. This increase occurs because of a reduction in the false positives as illustrated by an increase in the precision metric in Figure 2.7(a). Akin to session length, the F_1 measure is not significantly influenced by the small fluctuation in recall shown in Figure 2.7(b). This rise in the F_1 measure is very desirable because it suggests that the detector will continue to accurately detect Web robot sessions even as the proportion of Web robot traffic faced by a server continues to grow.

In summary, the performance of our approach in offline detection is very



(a) Precision (b) Recall



(c) F_1

Fig. 2.7: Evaluation metrics with varying session mixture

promising. For a test set containing an equal number of robot and human sessions, the F_1 measure of the detector approaches 0.85. Furthermore, the detector can accurately classify a session after analyzing only 12 requests. Finally, the performance improves with an increase in the proportion of robot traffic, suggesting that the approach will not cease to be effective as the level of robot traffic seen by Web servers continues to rise.

2.2.5 Real-time detection

In real-time detection, the detector is expected to classify a session as robot or human after analyzing only the first few requests. We believe that our approach lends itself well to such real-time detection because the F_1 measure converges after analyzing a few requests as seen in Section 2.2.4. In this section, we present the modified algorithm for real-time detection and analyze its sensitivity and performance.

Modified Algorithm

To adapt the detection algorithm for identifying sessions in real time, we introduce two new parameters k and Δ. k specifies the minimum number of requests that the detector must analyze before classifying a session. We note that while it is desirable that the detector classifies an active session early and by analyzing as few requests as possible, it is necessary to impose a minimum number to mitigate

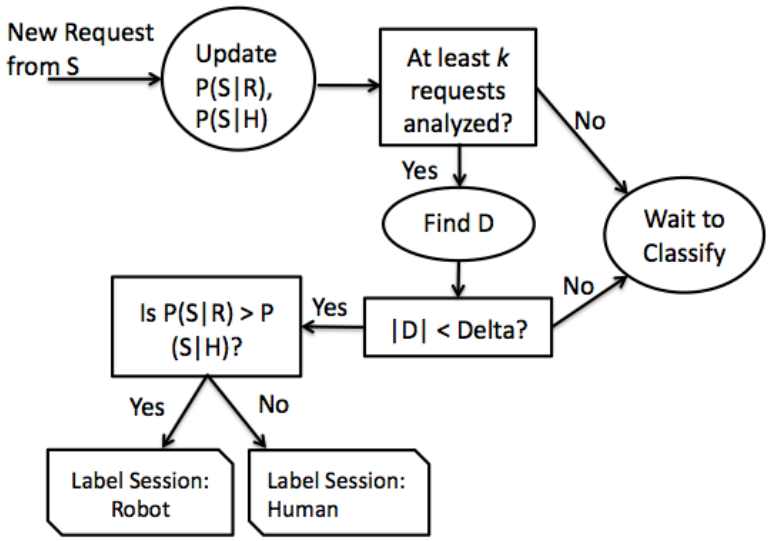


Fig. 2.8: Modified algorithm for real-time detection

the impact of fluctuations in the probabilites $Pr(\mathcal{S}|\mathbb{R})$ and $Pr(\mathcal{S}|\mathbb{H})$ likely during the first few requests in $\mathcal{S}$ on the classification decision. After at least k requests have been analyzed, if the difference in these log probabilities exceeds the second parameter Δ, the detector stops and classifies a session based on the model that has a higher probabilitiy. Otherwise, for every additional request, the detector continues to update the log probabilities until the difference is larger than Δ.

The above procedure used for classifying each session in real time, also shown in Figure 2.8, is summarized as follows:

1. Update $\log Pr(\mathcal{S}|\mathbb{R})$ and $\log Pr(\mathcal{S}|\mathbb{H})$ for every new request.
2. After at least k requests from $\mathcal{S}$ have been analyzed, whenever a new request is received compute $D = \log(Pr(\mathcal{S}|\mathbb{R})/Pr(\mathcal{S}|\mathbb{H}))$.
3. If $|D| < \Delta$, do not classify $\mathcal{S}$. Otherwise, if $D \geq 0$, classify $\mathcal{S}$ as a robot and

if $D < 0$ classify $\mathcal{S}$ as a human.

4. If a session ends before classification, use offline detection.

The values of both k and Δ control the granularity of the classification decision. For a small value of Δ, the detector can reach a classification decision earlier, allowing a differentiated treatment to robots sooner. However, premature classification decisions increase the risk of discriminating against legitimate human visitors. Therefore, to better understand the impact of Δ on the classification decision, we visualize the difference in the log probabilities $Pr(\mathcal{S}|\mathbb{R})$ and $Pr(\mathcal{S}|\mathbb{H})$ in Figure 2.9. We see that the difference in these probabilities stays similar and low over a wide range, and does not become appreciable until either $Pr(\mathcal{S}|\mathbb{R})$ or $Pr(\mathcal{S}|\mathbb{H})$ approach 1. Thus, if we set $\Delta > 2$, a session will be classified only if there is near certainty that it is generated either by $\mathbb{R}$ or $\mathbb{H}$ and not the other one. This will lead to a higher F_1 measure, albeit at the expense of not classifying many sessions. Thus, setting Δ in the range $0.5 \leq \Delta < 2$ will allow the detector to classify most sessions with broad degrees of confidence. We note that even after setting k and Δ to allow for classification with moderate certainty, the detector will still fail to classify those sessions with less than k requests or with request patterns that are not as distinct. Thus, in addition to the measures in Section 2.2.3, we use the percentage of unclassified sessions to evaluate the sensitivity of the real-time detector to the values of k and Δ.

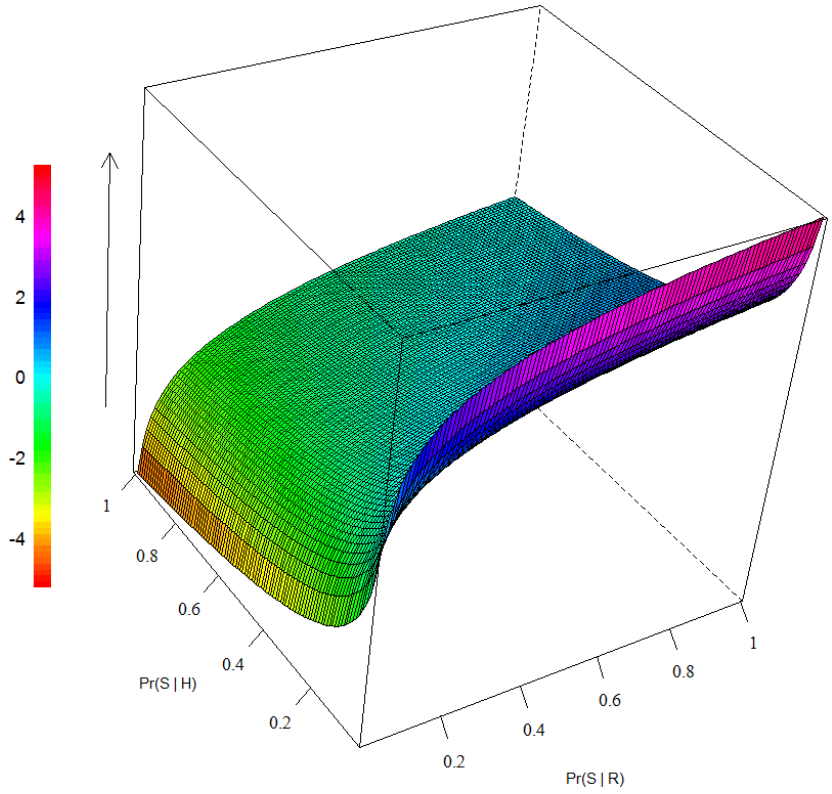


Fig. 2.9: Difference between the logs of $Pr(\mathcal{S}|\mathbb{R})$ and $Pr(\mathcal{S}|\mathbb{H})$

Sensitivity Analysis

To evaluate the sensitivity of the detector to parameters k and Δ, we replayed the arrival stream of robot and human requests used to evaluate the performance in offline detection in Section 2.2.4. We vary k in the range of 2 and 20 and Δ in the range of 0.5 and 2 as discussed in Section 2.2.5. Both parameters were varied simultaneously to expose cross-dependence, if any, between them. In this section, we present the results from the replay of the February 2008 data set. Measures for the other data sets exhibited similar trends and are not shown here.

Figure 2.10 visualizes the proportion of unclassified sessions as a function of k and Δ. As expected, the percentage of sessions that remain unclassified increase with Δ across all values of k. By the absolute measure, however, only 10% of the sessions remain unclassified for $\Delta \geq 1.85$ and $k > 3$. Moreover, this percentage decreases to approximately 5% starting at $\Delta = 1.4$. Even though only a small percentage of sessions remain unclassified at $\Delta = 1.4$, our confidence in the classification decision remains high because the probabilities $P(\mathcal{S}|\mathbb{R})$ and $P(\mathcal{S}|\mathbb{H})$ need to be very different for the detector to label a session. For example, if $P(\mathcal{S}|\mathbb{R}) = 0.7$, $\Delta = 1.4$ requires that $P(\mathcal{S}|\mathbb{H}) \leq 0.172$ before we can classify $\mathcal{S}$ as a robot. Thus, the real-time detector is able to classify sessions with reasonable confidence while allowing just a small percentage of sessions to remain unclassified.

We examine recall, precision and F_1 as a function of Δ and k for the February 2008 test set. Figure 2.11 shows that precision varies widely if we set $k < 12$ for

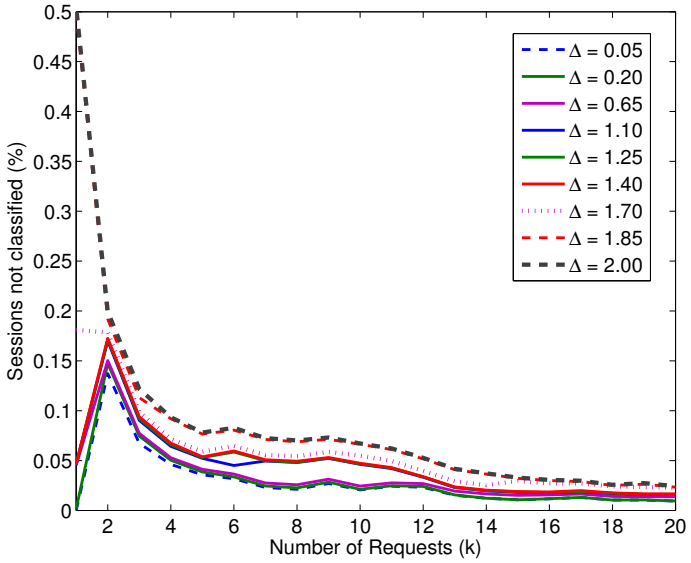
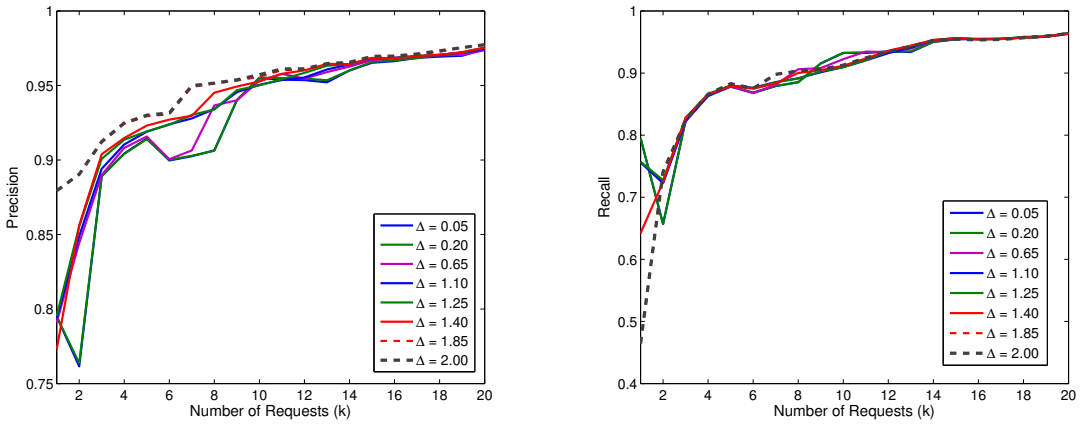


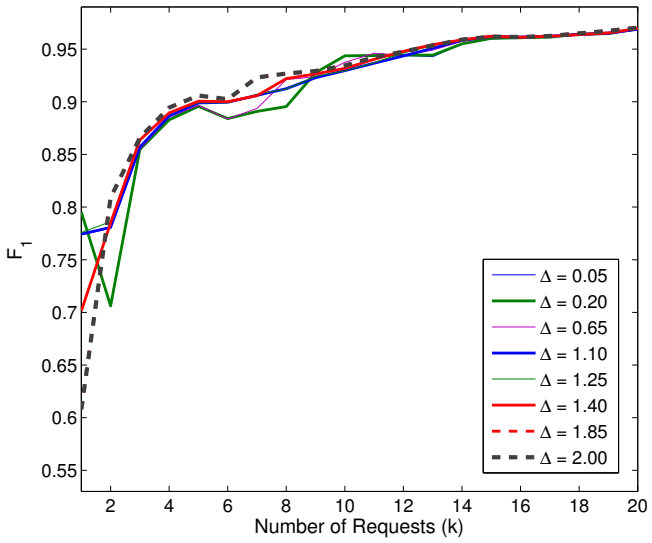
Fig. 2.10: Percent of unclassified sessions

low values of Δ. This is consistent with the performance of the offline detector in Figure 2.6, where the F_1 metric stabilized only after 12 requests. For $\Delta \leq 1.40$, the changes in precision when $k \leq 10$ is significant, as large values of Δ preclude some sessions from being classified before they are terminated. For example, if we let $k = 6$, precision rises by approximately 0.03 as Δ increases. When k is greater than 10, however, the influence of Δ begins to wane. This is because after examining the first 10 requests for most sessions, the difference between $Pr(\mathcal{S}|\mathbb{R})$ and $Pr(\mathcal{S}|\mathbb{H})$ becomes so significant that only very large values of Δ will preclude the session from being classified. When $k \geq 10$, we achieve a very low false positive rate (approximately 5%), as precision reaches nearly 0.95 for $\Delta \geq 1.4$.

Figure 2.11 examines recall as a function of k and Δ. Unlike precision,



(a) Precision (b) Recall



(c) F_1

Fig. 2.11: Performance metrics for varying k and Δ

which shows no clear trend across different values of Δ, recall changes similarly as a function of k for different values of Δ. Recall is also more sensitive to k compared to precision, as it increases by approximately 0.10 as $3 \leq k \leq 12$. Figure 2.11(c) shows F_1 as a function of Δ and k. For $k \geq 4$, F_1 remains above 0.85, is not strongly influenced by the value of Δ, and gradually increases to an extremely promising $F_1 = 0.95$ at $k > 13$.

In summary, we find that our real-time detection approach can achieve very high performance ($F_1 = 0.95$) with a very low false positive rate, while letting less than 5% of all sessions remain unclassified when we use $\Delta = 1.7$ and $k > 13$.

2.3 Chapter Summary

This chapter presented a comprehensive survey and a novel classification for contemporary Web robot detection techniques, and presented a new approach for detecting Web robots that addresses the current limitations in the state-of-the-art. Through a critical analysis and comparison of the categories that detection techniques fall under, we argued that analytical learning techniques have the most potential to provide robust Web robot detection. To address the key limitation of such approaches, namely, their dependency on learning traffic features that may change as Web robot traffic evolves over time, our approach is based on learning the differences within the resource request patterns of robots and humans, which

is a fundamental distinction that is unlikely to change over time. Out experimental results indicated that our detection approach is accurate and reliable, both in the offline and real-time mode.

Chapter 3

Robot Classification

Although many Web robots may visit a Web server to advance a noteworthy cause, like information retrieval for news aggregation services and web search indexing, others may be ill-behaved and/or malicious and may exhibit undesirable behavior including:

1. not following the operational limits specified by robots.txt or by standards that define ethical Web robot behavior [109];
2. spying on or censoring the flow of information; and
3. visiting sites to copy html code and identify ways to attack Web servers

The crawls of Web robots that exhibit such behaviors should ideally be curtailed, or even blocked; in the best case, they consume precious resources and in the worst case they may compromise the security, functionality, privacy, and performance of a Web server. To enable server administrators to curtail the activities of undesirable robots, a method that can quickly distinguish between well-behaved, benign robots and ill-behaved, malicious ones is essential. To facilitate such dis-

tinction, it is first necessary to understand the key properties of their crawling behavior, classify them based on these crawling properties, and then infer the purpose of their visits based on this classification. While it may not be feasible to definitively infer the intent of a robot and its creator from such classification, such partitioning may nevertheless provide significant insights into whether a robot's behavior follows desirable or undesirable trends. The development of a taxonomy for Web robots also provides a framework for cataloguing and understanding the wide variety of contemporary Web robots that have arisen out of new kinds of Web services and seek to collect dynamic, time-sensitive opinions and information on the Web [16,120,121].

In this chapter, we classify and then interpret the types of Web robots that constitute contemporary streams of Web traffic. Our novel classification scheme using three orthogonal perspectives: first via the lens of a robot's stated functionality, then by examining the resources it targets, and finally by the workload their visits impose on the Web server. Each viewpoint allows us to observe the properties of a Web robot that would be invisible from the alternate perspective. Furthermore, the observations across these views can be leveraged to determine the likelihood that a robot is ill-behaved or has malicious intent. The classification framework is applied to Web robots identified across an entire year of requests to the UConn SoE Web server[1].

[1]Portions of this chapter were previously written and published by the author in [31,32,38,33].

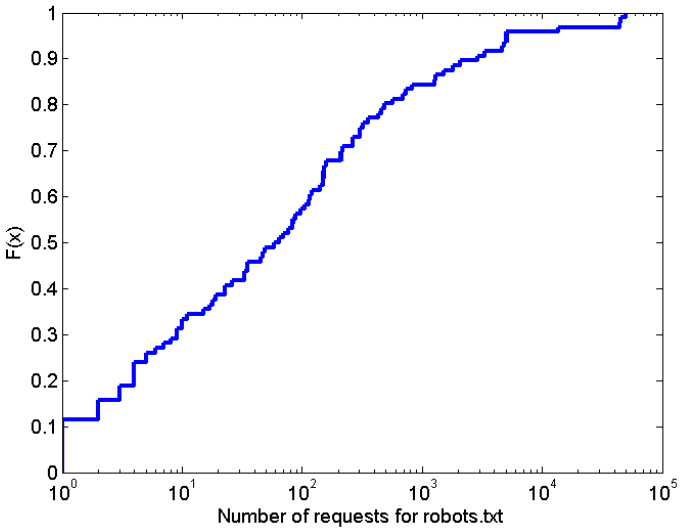


Fig. 3.1: Distribution of the number of requests for robots.txt

This chapter is organized as follows. Section 3.1 gives a preliminary analysis of Web robot requests on the SoE Web server. Sections 3.2, 3.3, and 3.4 presents and applies the functional, resource, and characteristic dimensions of our classification scheme, respectively, over Web robot sessions.

3.1 Preliminary Analysis

To motivate the need for a comprehensive robot classification scheme, we first present a preliminary analysis of Web robot requests and demonstrate that they are highly varying. This preliminary analysis is based on traffic collected on the UConn SoE Web server between February 2007 and January 2008.

3.1.1 Robot requests for robots.txt

Our analysis indcated that only 78 of all robots found (58%) requested the file *robots.txt*, which gives the robot instructions as to what resources should and should not be visited during their crawl. Figure 3.1 plots the shows the cumulative distribution function (CDF) of the number of requests for the file *robots.txt*. Approximately half of the 78 robots requested the file less than 100 times, whereas the request count for the other half varies widely from 100 to 10,000. The CDF exhibits a linear trend when the number of requests is plotted on a log scale. This means the request counts are evenly distributed across a large range, making it difficult to determine a representative range of frequencies with which a majority of the robots will request this file. Nearly 40% of the robots choose to completely ignore these guidelines, and those that do robots.txt are only partially interested in following the guidelines specified within.

3.1.2 Robot HTTP requests

Figure 3.2 shows a log plot of the number of HTTP requests from robots and from all users (including robots) received by the server. We see that the total month-to-month traffic decreases from May through August. During this period classes were not in session, due to which only a few students needed access to SoE sites. Traffic returns to pre-May levels in September, when classes resume. Although the total number of requests seen by the server follows this pattern,

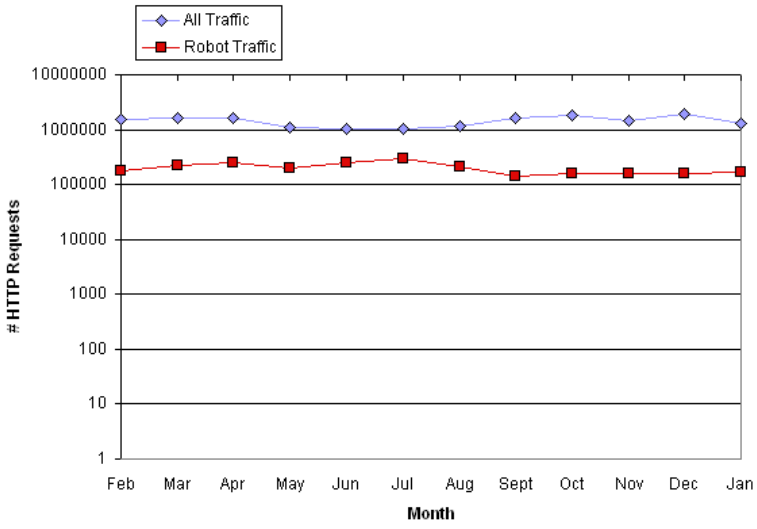


Fig. 3.2: HTTP requests from robots and all users

the number of HTTP requests sent by robots does not exhibit any pattern. In fact, there is no correlation between the number of requests sent by robots to the month or to the overall traffic presented. These findings are consistent with the previous work by Lu et. al. which discusses how Web robots are not aware of the current server workload [126]. Thus, Web robots disregard the seasonal busy periods of human traffic based on the domain of the server (for example, low workload during the summer for academic servers or a high workload during the holiday season for e -commerce servers) and continue to send requests ignoring these predictable workload patterns.

	Robot Traffic	All Traffic
HTTP Requests	3,198,530	17,293,248
Bandwidth Consumed	328.80 GB	4,185.98 GB

Table 3.1: Request and bandwidth consumption, Jan 2007 - Feb 2008

3.1.3 Robot traffic intensity

Table 3.1 reveals the total bandwidth and number of requests received by the server over the entire period. Robots sent approximately 18.49% of all the requests, and consumed around 7.68% of the bandwidth. The proportion of bandwidth attributed to robots may be conservative because of the domain of the server – it is not uncommon for faculty, staff, and students connected on the same LAN as the SoE server to download large documents, presentations, and disc images of software. As a result, the bandwidth consumption of humans may be inflated relative to the bandwidth consumed by the robots. Furthermore, constant backups of data across the network cause human users to consume additional bandwidth. Thus, although the internal traffic drives down the relative proportion of bandwidth consumed by robots, it is possible that this consumption, as a proportion of external traffic, may be significant because approximately one in five requests originates from Web robots.

This preliminary analysis suggests that Web robots exhibit different levels of commitment in following the crawling guidelines specified in *robots.txt*, represent a

significant percentage of the traffic handled by a Web server, consume a significant amount of bandwidth, and are agnostic to the busy periods of a Web server. To better understand and then prepare Web servers for handling their requests, it is necessary to classify robots into groups that exhibit similar crawling behaviors and demand characteristics.

3.2 Functional Classification

In the functional classification scheme, we partition Web robots into classes according to their functions. To enable this classification, we followed a two-step process to research the functionality of the 134 robots identified from the UConn SoE log by the tool *AWStats* [9]. In the first step, we determined if the user-agent field of the HTTP request contained a URL. A well-behaved robot uses a Web site to identify itself and to explain its functionality, and *AWStats* provides links to these URLs. When a URL did not exist, we manually searched the Internet to determine the function of the robot. For most robots, we were able to either verify their function based on a Web site developed by the robot's authors, Web sites that list the functionality of common robots, or sites that host discussions among server administrators about its functionality. If a robot's function could not be ascertained using this two-step process, we classified it as unknown.

If we classified a robot's function based on the discussions among the administrators, we verified that what we observed from these discussions was consistent

Type	Name
I	Indexer
E	Experimental
V	Verifier
R	RSS Crawler
A	Analyzer
H	Harvester
S	Scraper
IRC	IRC
$Anon$	Anonymous
U	Unknown

Table 3.2: Robot functional types and names

with the actual behavior of the robot. We verified this consistency by ensuring that the robot came from the same host, requested the same type of resources, and exhibited similar session characteristics. Based on this analysis, we consider the following function categories to classify robots.

- **Indexer (I)** - This is the most general function class of all Web robots.
- **Experimental (E)** - A robot that is designed primordially for research or experimental purposes. Research about the origin and background of the robot points to this classification.
- **Verifier (V)** - A robot that traverses a site to verify that a Web page does not contain broken links and that all resources intended to be accessible.
- **RSS Crawler (R)** - A robot that only retrieves information from the RSS feeds of a Web site or a blog.
- **Harvester (H)** - A Web robot that sends HTTP requests for items other

than HTML documents and their embedded resources. For example, a robot which sends requests for all music, video, and/or picture files from a Web page.

- **Analyzer (A)** - A type of harvester that downloads Web resources only for analytic, archival, or scientific purposes. Analyzers have documented functionality and are assumed to use the collected resources in a safe and predictable manner.
- **Scraper (S)** - A robot that saves the requested HTML documents. The term *scraper* is jargon to describe such robots, as they are commonly used to automatically create copies of Web sites for malicious purposes.
- **IRCBot (IRC)** - A robot that connects to an IRC server.
- **Anonymous (Anon)** - A robot which does not define itself in the user-agent field of a request. Such robots are intentionally designed to remain anonymous.
- **Unknown (U)** - A robot whose function is private, undocumented, or unknown.

Very often a robot exhibits multiple functions due to which it may be placed in several classes. For example, an indexer that also saves all jpg files to a centralized database could be a class I or a class H robot. As a second example, while an experimental robot performs some analysis on the collected resources, classifying such a robot both as an experimental (E robot) and an analyzer (A

robot) makes its primary function ambiguous and open to interpretation. Despite the analysis that an experimental robot may conduct, it is important to classify it as an E robot and not an A robot, because an E robot may exhibit ill-behaved functionality, while A robots are assumed to operate in a well-behaved fashion. Such a single assignment makes the main functionality of a robot clearer so that it can become the focus of our analysis. In order to classify each robot into a single function class, we developed a hierarchy to properly classify multi-function robots. This hierarchy is structured according to three criteria. Each criterion, and the order in which it should be applied, is organized so that a robot will always be placed into a class that is most critical for determining its desirability. The criteria and their order are as follows:

1. **Undocumented functionality:** Undocumented robots are intuitively high risk, as they are likely to be ill-constructed or intentionally undocumented.
2. **Specialty:** This criteria is important to distinguish robots that are more specialized than the others. For example, a S robot is also a H robot, but because a S robot is defined more specifically, those robots fall into this class.
3. **Variety and size of resources:** According to this criteria, robots whose functionality requires them to request multiple types of resources are ranked higher than those that must request only a singe type of resource. If the types of resources requested by two function classes are similar then the

function class that requests resources with sizes that are on an average larger is ranked higher.

Undocumented or anonymous robots must be identified before others because these may be extremely hazardous due to their non-deterministic behavior. The next criterion, specialty, will ensure that a robot will be classified as specifically as possible, because such precise classification can provide valuable insights into its desirability. Finally, the size and variety of resources are considered, as robots whose functions cause them to request unpredictable or large resources should be highlighted. Using the above guidelines, we order the function classes into the following hierarchy:

$$Anon > U > E > A > S > H > R > I > V > IRC$$

If a robot shows behavior that fits both function classes α and β, and if $\alpha > \beta$ then the robot will be classified as α. This hierarchy can be applied naturally to robots that fit multiple function classes.

Anon and U robots are placed highest in the hierarchy following the first criterion. E robots can retrieve a large variety of resources of various sizes depending on the experiment they are designed to run. Because of the unpredictability of the resources collected and also because the purpose of such experimental robots may not always be clearly defined, class E is placed just below class U. A robots, like E, collect a large variety of resources with highly variable sizes depending on

the analysis. $E > A$, however, because a type E robot is a type A robot but uses the data in a scientific manner that is not always documented, whereas a type A robot uses the data in a predictable or a standard manner (for example, to collect aggregate statistics about a Web server for public or private use). Following the same line of reasoning, class S is placed higher than class H. Class I follows as a type of a robot that primarily gathers and parses the HTML documents, retrieves links, and then discards the HTML. As a result, a class I robot requests predictable resources that are small. Class R precedes class I because an RSS Crawler acts just like a class I robot but on a smaller, more specialized set of Web resources. Class V robots are placed accordingly, as they only crawl a site to verify that no links are broken. Because class V robots should never request resources other than HTML pages, the average size of the resources they request is small. Verifiers then use the HEAD HTTP command to verify that links and resources still exist. IRC robots are at the bottom of the hierarchy, as these should never appear in a server access log unless an IRC server is also in place. Anchored around this function class hierarchy, we now present a refined analysis of the aggregate statistics collected by *AWStats*.

Percentage of HTTP requests

Figure 3.3 shows the breakdown of the percentage of HTTP requests according to the function classes. The figure indicates that a large percentage of HTTP

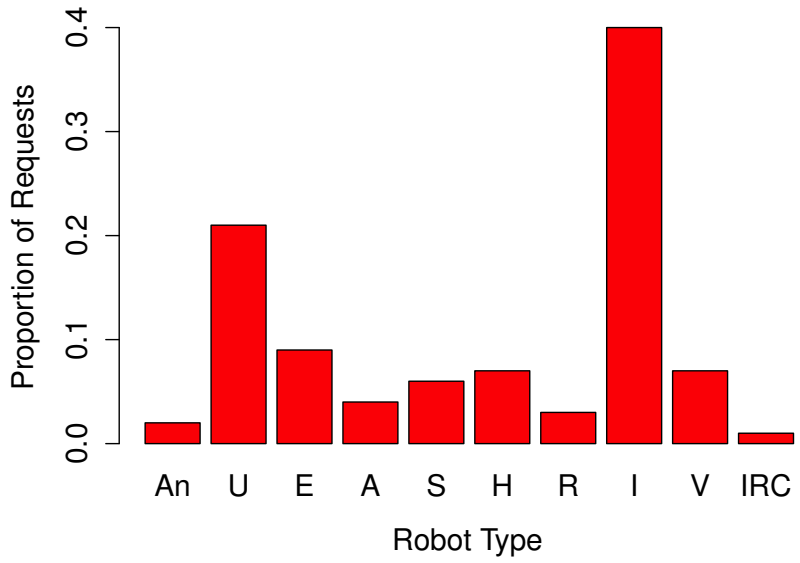


Fig. 3.3: Percentage of robot HTTP requests per function class

requests are from I robots. This is reasonable as it encompasses all general purpose indexers (the role most often taken by a Web robot) that do not meet the specific criteria of an H, S, A, or E robot. Combining the statistics of U and *Anon*, it is startling to find that approximately 23% of the HTTP requests are presented by robots with unknown functionality.

Bandwidth consumption

Figure 3.4 compares the bandwidth consumption of each function class. As expected, the bandwidth consumption of I robots is the highest, consistent with the highest percentage of HTTP requests. Surprisingly, the bandwidth consumption of *Anon* robots is comparable to H robots, although the percentage of HTTP re-

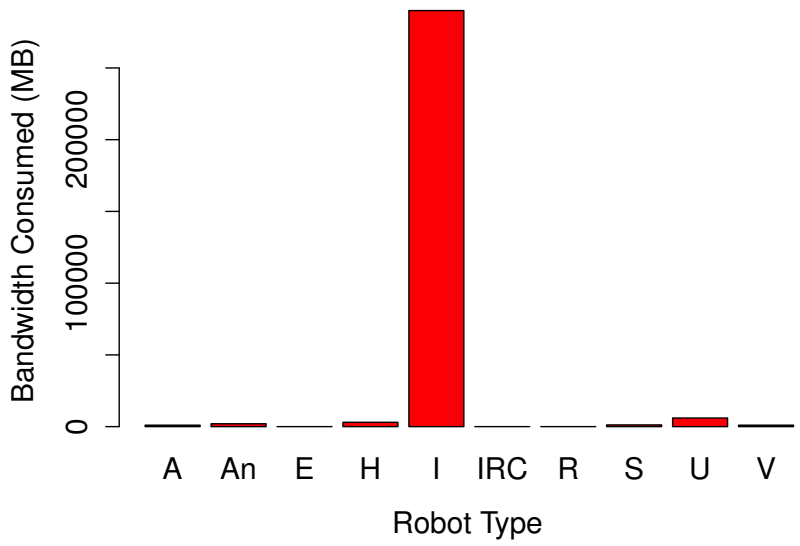


Fig. 3.4: Bandwidth consumption per function class

quests from H robots is much higher than the percentage of requests from *Anon* robots as seen in Figure 3.3. Because *Anon* robots have non-deterministic behavior, they should be closely inspected to better understand this phenomenon.

Size of requested resources

In Figure 3.5, the average size of the resources requested by robots in each function class is presented. It can be seen that the average size for classes which request resources other than HTML pages (A, S, H) is larger than the size of the resources requested by class I robots, which overwhelmingly request HTML documents. U robots also request large resources on an average. Since their function is undocumented or proprietary, and they request a relatively large volume

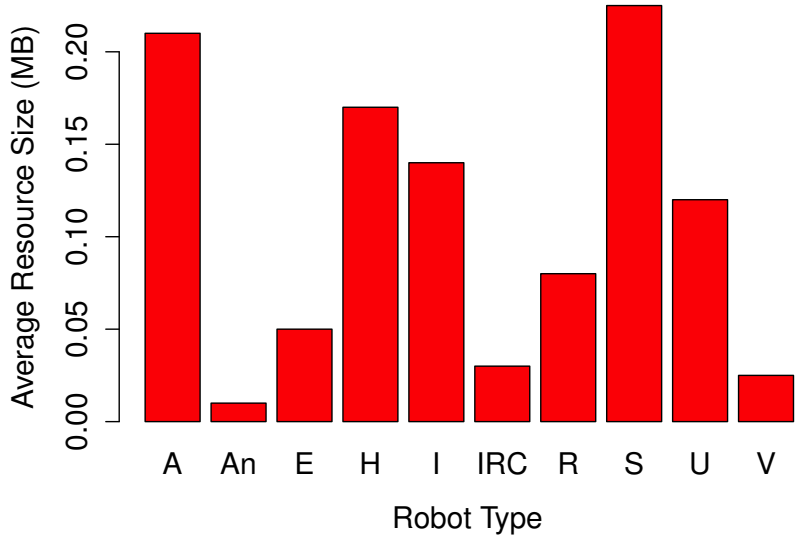


Fig. 3.5: Average size of requested resources per function class

of data, it further raises the doubt that robots from class U may be undesirable.

Indexer robots also request resources that are larger than expected. According to Levering *et. al.*, the size and composition of an average HTML document is 25 KB [76], so we expect to see indexers request resources that are approximately of this size. Since our class of indexers covers a large variety of robots, including poorly developed and multimedia indexers, however, it is likely that some indexers also request resources besides HTML pages for analysis. We should also consider the academic domain of this server, where faculty Web pages are small and simple, but rife with links to .pdf and .ps documents that are generally much larger than 25 KB.

In conclusion, we have established how the functional classification can be used to study the influence of a robot's function on its crawling properties. This classification helped us identify the following trends:

- 23% of robots have unknown or undocumented functionality.
- 40% of all robot requests originate from indexers.
- The average size of a requested resource is not estimated by the average size of an HTML page.

3.3 Resource Classification

The second dimension of classifying Web robots considers their preferences for the different types of resources. For this classification, we consider a subset of the resources used in our Web robot detection algorithm, namely:

- **Web (web)** - Web resources include any Web page stored on the server. *Web* resources are requested by all types of robots to learn about the structure of the site for future crawls. These file types include htm, html, php, cgi, and shtml.
- **Text (txt)** - These include Web resources that are encoded in some text-based format. Such resources may be parsed by Web robots for analysis, or archived for some purpose. Example file types in this class include txt, xml, cs, cpp, java, css, m, and *robots.txt*.

- **Image (img)** - These resources comprise image data from a server. Examples include jpg, eps, ico, bmp, raw, svg, and eps.
- **Audio and visual multimedia (av)** - These resources include multimedia data from a Web server, excluding images. Examples include avi, midi, mp3, wmv, mpg, aif, wav, and rm.
- **Document (doc)** - These consist of resources that use a formatted document file type. These differ from *txt* resources because they are encoded to be read by a specific application. Examples include doc, dvi, db, pdf, ps, and ppt.
- **Other (oth)** - This class includes resources that do not belong to any one of the above classes. Blank http requests, file types with no extension, compressed files, binary files, or extensions not covered above fall into this class.

It is essential to capture requests to *txt*, *img*, *doc*, and *av* resources since these are the traditional targets of Web robots. While *web* resources also fit the definition of *txt* type resources, we include these in a class of their own. If *web* and *txt* resources were combined into one class, it would be infeasible to determine the frequency of requests for *txt* and *web* resources separately. Assessing the frequencies of requests for these two types of resources separately, however, is important because while all robots request *web* resources to learn the structure of the site and to find specific resources, some robots specifically target *txt* resources and do

not favor *web* resources. As an example, the robot *Google Feedfetcher* requested $2,358$ *txt* resources but only 20 *web* resources. This is because *Feedfetcher* is an RSS crawler, grabbing xml formatted RSS feeds from the Web server. Resources in class *oth* such as binaries, Java class files, files without extension, etc. are useful to index for search engines and information repositories, however, such resources cannot be analyzed meaningfully without some context. As a result, identifying non-indexing robots which regularly request *oth* types of resources may suggest ill-behavior and unnecessary resource consumption.

AWStats provides only aggregate resources consumed by the robots, and not fine-grained information about the specific types of resources requested. Therefore, we wrote a custom analyzer to process the access log to identify robots and the specific resources each one requests. The custom analyzer uses the same database of regular expressions to identify robots as *AWStats*. For each robot, it generates a list of all resources requested using the HTTP methods GET, HEAD, and POST. Using this information, we then extracted the distribution of requests for each type of resource. We define the metric *favoritism index*, to quantitatively determine the degree to which a robot favors each type of resource. The favoritism index is defined as follows: for a robot R requesting resources of type T, the favoritism index is given by $r_f(R,T) = R_T / \sum_i R_{T_i}$, where R_T is the number of requests from robot R for resource type T, $\sum_i R_{T_i}$ is the total number of requests sent, and $\sum_i r_f(R,T_i) = 1$ for each robot R. The resource type for which r_f is

the highest is considered as the favorite type of resource for the robot.

We compute the favoritism index of each robot for each type of resource. Using these values, we then group robots according to their highest favoritism index. For example, robots who have the highest favoritism index for the *img* type of resources are placed into the *img* class. In the sequel, we refer to robots that favor resource type A, as "A robots", for example, robots which have the highest favoritism index for *txt* type resources are grouped into the *txt* class and are referred to as *txt* robots. Next, we analyze the behavior of Web robots based on this resource classification scheme.

Resource favoritism across all robots

Table 3.3 shows the favoritism indices for a sample of Web robots, which illustrates the variety of favoritism indices that robots may exhibit. *Grub.org* sent requests for several types of resources, but favored *web* resources the most. *Shim Crawler*, on the other hand, only requested *web* and *txt* documents, favoring the latter. While *TencentTraveler* and *Gaisbot* both favor *img* resources, the degree of favoritism of *TencentTraveler* is significantly stronger than *Ultraseek*. Furthermore, *LinkChecker* and *Checkbot* exhibit strong favoritism indices for *oth* resources. A robot such as *Checkbot*, whose favoritism index for *oth* is 1, may be behaving suspiciously. By targeting *oth* resources exclusively, *Checkbot* raises the suspicion that it has gained knowledge about the structure of the site through some other

Robot	web	txt	img	doc	av	oth
Gaisbot	0.3540	0.2795	0.3665	0	0	0
Grub.org	0.8993	0.0158	0.0676	0.0083	0.0023	0.0067
Shim Crawler	0.3333	0.6667	0	0	0	0
TencentTraveler	0.0799	0.0801	0.7955	0.0292	0.0139	0.0011
Ultraseek	0.3855	0.1053	0.0023	0.4776	0.0005	0.0288
LinkChecker	0.0449	0.0449	0	0.0393	0	0.8708
Checkbot	0	0	0	0	0	1

Table 3.3: Favoritism indices for sample robots

means, and is only requesting those resources that traditional robots would not be able to analyze.

Figure 3.6 shows the distribution of all the robots into different resource categories. As expected, the *web* category dominates, since robots need to crawl HTML pages to find links to their targeted resources. Robots are distributed relatively uniformly across the remaining resource classes, with the exception of class *av*. Our academic SoE server does not host any audio and video files, due to which there are no robots that favor these types of resources. The distribution also suggests that our proposed resource classification scheme reasonably models the behavior of Web robots on this server from the perspective of resource consumption, and distributes robots with high fidelity according to their preferred types of resources.

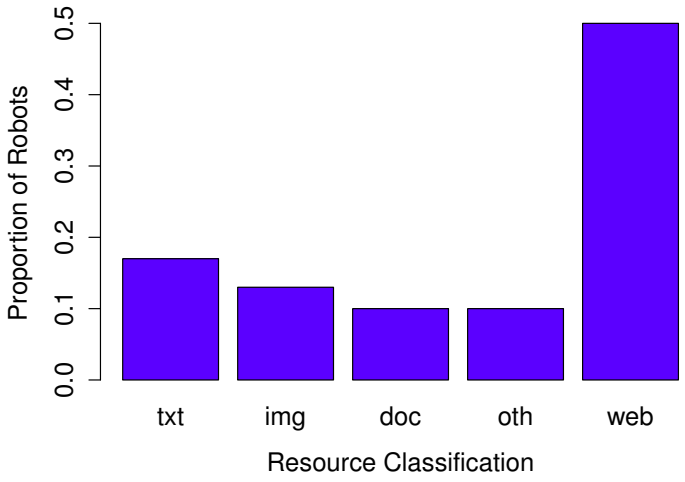


Fig. 3.6: Distribution of robots into resource classes

Resource favoritism by resource class

Next, we compare the favoritism indices of the robots belonging to each resource class, using the indices of the robots in the *img* and *web* categories for illustration. The favoritism indices for a sample of *img* robots, given in Table 3.4, suggest that *img* robots very strongly prefer image files, as only four robots have a favoritism index of less than 0.68 for img resources. Furthermore, *img* robots exhibit very different favoritism values for the remaining types of resources. This wide variation could be due to many reasons, including the total number of requests sent, and the amount of prior information the robots had about the content on the site. If a robot were to know the location of *img* files before starting its crawl, it would not need to send requests for *web* documents to search for possible links. For example, 84% of the requests from the robot *Motor* are for image files, and only 5% are

Robot	web	txt	img	doc	av	oth
Aport	0	0	1	0	0	0
Bloglines	0	0	1	0	0	0
Boris	0.0135	0	0.9864	0	0	0
ConveraMMCrawler	0	0.1600	0.6800	0	0	0.1600
Cursor	0.1071	0.1071	0.7142	0.07142	0	0
Custo	0.0974	0.0720	0.7309	0.09216	0.0063	0.0010
Exabot	0.0791	0.0669	0.8508	0	0	0.0030
HTTTrack	0.2025	0.0498	0.3633	0.3505	0.0115	0.0222
Motor	0.0526	0.0526	0.8421	0.0526	0	0
MSIECrawler	0.1305	0.2975	0.5316	0.0287	0	0.0115
Scooter	0.3746	0.0450	0.5509	0.0062	0.0230	0
Steeler	0.3214	0.1904	0.3928	0.0952	0	0
Sunrise	0	0	1	0	0	0
TencentTraveler	0.0799	0.0801	0.7955	0.0292	0.0139	0.0011
WebCollage	0.2249	0	0.7750	0	0	0
Webdup	0	0.25	0.75	0	0	0
Yahoo-MMCrawler	0.0003	0.004	0.9948	0	0	0.0007

Table 3.4: Favoritism indices for robots in class *img*

for HTML pages. It is not realistic to overwhelmingly request so many image files while sending very few requests for *web* resources without having some offline knowledge about the structure of the site, or the location of various resources.

Table 3.5 shows the favoritism indices for all types of resources for a sample of robots in the *web* resource class. In this class, the values of the index are more uniformly distributed across the different types of resources compared to the *img* class of robots. This is because all robots, regardless of their resource class, must request *web* documents in order to find their targeted resources. This hypothesis is supported by observing that robots in the *web* class also exhibit comparatively higher preference for the remaining types of resources in addition to *web* resources.

For example, the robots *BaiDuSpider* and *Combine System* exhibit a favoritism index of approximately 0.40 for *txt* resources. Similarly, *Gaisbot*'s favoritism index for *txt* and *img* resources is approximately 0.28 and 0.37, respectively. By comparison, robots in the *img* resource class show only a mild favoritism for *web* resources as seen in Table 3.4.

Table 3.6 reports the mean and the variance of the favoritism index of the preferred resource type for each resource class. Referring to the table, we observe that robots belonging to the *web* class have a mean favoritism index for *web* resources of 0.6903. The average favoritism index of *oth* robots for *oth* types of resources is the highest. This means that these robots send requests for *oth* resources at a significantly higher rate than the robots in any other resource class. While it may be reasonable that malfunctioning or ill-designed robots may send many requests for *oth* resources, it is against intuition to see robots favoring *oth* resources more strongly than well-defined defined robots that are built to request specific types of resources. One reason for this may be that *oth* robots may perform widely different functions as opposed to robots in the remaining resource classes.

Since statistics are very similar across *web*, *txt*, and *doc* classes, we conjecture that many robots have offline knowledge about the locations of the resources they desire before beginning a crawl. Were this not the case, we would have seen

Robot	web	txt	img	doc	av	oth
Asterias	0.5073	0.1580	0.0006	0.0006	0.3305	0.0030
BaiDuSpider	0.4180	0.3951	0	0.0276	0.0119	0.1474
boitho.com-dc	0.9588	0.0394	0	0	0	0.0018
Combine System	0.5772	0.3960	0	0.0134	0	0.0134
ConveraCrawler	0.7750	0.0156	0.0002	0.1952	0	0.0139
Digger	0.5000	0.5000	0	0	0	0
DoCoMo	0.5517	0.2069	0.1034	0.0517	0	0.0862
EasyDL	0.9677	0	0	0.0323	0	0
Emacs-w3 Search Engine	1.0000	0	0	0	0	0
ExactSeek Crawler	0.6875	0.3125	0	0	0	0
Findlinks	0.5236	0.4764	0	0	0	0
Gaisbot	0.3540	0.2795	0.3665	0	0	0
Geniebot	0.7826	0.2018	0	0.0130	0	0.0026
GetBot	0.5000	0.5000	0	0	0	0
GoForIt.com	0.6250	0.3750	0	0	0	0
Google Sitemaps	0.5117	0.0429	0.0713	0.3193	0.0067	0.0481
Googlebot	1	0	0	0	0	0
Grub.org	0.8993	0.0158	0.0676	0.0083	0.0023	0.0068
OmniExplorer Bot	0.9712	0.0229	0	0.0030	0	0.0029
RoboCrawl Spider	0.8193	0.0390	0.0005	0.0815	0.0305	0.0292
Robozilla	1.0000	0	0	0	0	0
SlySearch	1.0000	0	0	0	0	0
SuperBot	0.6667	0	0	0.3333	0	0
SynooBot	0.6462	0.3538	0	0	0	0
Turn It In	0.5814	0.0374	0.0191	0.3073	0	0.0548
UdmSearch	0.9412	0.0588	0	0	0	0
Voyager	0.8569	0.1383	0.0037	0	0	0.0011
VSE	0.6667	0.3333	0	0	0	0
yacy	0.6134	0.3697	0	0.0168	0	0
Yahoo Feed Seeker	0.5907	0.0694	0.0011	0.3161	0.0022	0.0204
Yahoo Slurp	1.0000	0	0	0	0	0
Yandex bot	0.6016	0.0319	0.0036	0.3422	0.0014	0.0193
Zeus Webster Pro	0.5000	0.5000	0	0	0	0
Fast-Webcrawler	0.5442	0.0195	0.0002	0.4228	0.0020	0.0113

Table 3.5: Sample favoritism indices for robots in class *web*

	web	txt	img	doc	av	oth
Mean	0.6903	0.6971	0.7622	0.7147	0	0.7927
Variance	0.2875	0.2959	0.3068	0.2918	0	0.3233

Table 3.6: Mean favoritism index for resource classes

robots favoring *web* resources with a higher favoritism index compared to the other resource types, because most robots would need to traverse a great deal of html pages on a web site to reveal the other resource types hosted.

In summary, classifying Web robots based on the types of resources requested allowed us to observe that:

- About half of all robots favor *web* resources.
- Robots favoring *img* resources overwhelmingly prefer them.
- *web*, *txt*, and *doc* robots prefer their favorite resource type similarly.
- Robots favoring *oth* resources do so more strongly compared to robots in other resource classifications.
- Robots not heavily favoring *web* resources are likely to have some pre-existing knowledge about the structure of the site.

3.4 Characteristic Classification

In the final dimension of our classification scheme, we partition robots according to the workload characteristics they exhibit on a server. We demonstrate the feasibility of using K-means clustering for this purpose, by applying it to robots extracted

from UConn SoE server access logs. K-means is a common algorithm that has been used to analyze and partition data in many different domains [53,128,43]. We choose K-means clustering to partition Web robots because of its past successes in analyzing Web server requests [77,103].

To cluster Web robots, it is necessary to define an appropriate distance metric between data points. The selected metric must factor in the likely correlation between observations used to characterize robot traffic; for example the volume of http requests and number of bytes transferred may be correlated [77]. Furthermore, it should also consider that the observations may be measured across different scales; for example inter-arrival times between requests may be measured in seconds, and the average number of requests sent per session, could be measured as a count. We use the Mahalanobis distance, which incorporates both of these considerations, to cluster robots. The Mahalanobis distance between two feature vectors $vecx$ and $\vec{y}$, whose components represent a property of robot traffic, is defined as:

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \Sigma^{-1} (\vec{x} - \vec{y})}$$

where Σ is the covariance matrix for all observations and the superscript T denotes the transpose.

K-means clustering requires that the number of clusters k be selected before clustering commences. Each application of the algorithm is guaranteed to

have k clusters, so different values of k will lead to a unique clustering result. Thus, the value of k governs the quality of clustering, making its selection crucial. Because our objective is to partition Web robots so that all the robots in a group will display similar crawling characteristics, we consider two important criteria in selecting the value of k. The first criterion is concerned with minimizing distance within clusters (intra-cluster distance) while maximizing the distance between clusters (inter-cluster distance). Intra-cluster distance is defined as the distance from a vector to the centroid of the cluster to which it is assigned, while inter-cluster distance is defined as the distance from a vector to another one that does not belong to its cluster. Intuitively, the best clustering will be one that maximizes the inter-cluster distance and minimizes the intra-cluster distance. We measure the first criterion using the silhouette coefficient [113] metric, defined as follows: let $\hat{C} = \{C_1, C_2, ...C_k\}$ be the result of a clustering, fully partitioning a set of data points D. Define the distance of a data point $d \in D$ to some cluster $C_i \in \hat{C}$ as

$$dist(d, C_i) = \frac{\sum_{d_i \in C_i} dm(d, d_i)}{|C_i|}$$

where dm is the distance function between points. Let $\alpha(d) = dist(d, C_i^*), d \in C_i^*$ be the distance from d to its assigned cluster C_i^* (i.e. measuring intra-cluster distance) and $\beta(d) = \mathbf{min}_{C_i \in \hat{C}, C_i \neq C_i^*} dist(d, C_i)$ be the distance from d to the nearest cluster d is not assigned to (i.e. measuring inter-cluster distance). The *silhouette* of d is defined as:

$$\phi(d) = \frac{\beta(d) - \alpha(d)}{\mathbf{max}(\beta(d), \alpha(d))}.$$

$\phi(d)$ will approach -1 as the inter-cluster distance decreases and intra-cluster distance increases, and will approach 1 in the mirroring case. Thus, the closer $\phi(d)$ is to 1, the better the cluster assignment for d is. The silhouette coefficient of a clustering is simply the average value of the measure for each data point d:

$$SC_{\hat{C}} = \frac{\sum_{d \in D} \phi(d)}{|D|}$$

Previous studies suggest that values of $SC_{\hat{C}}$ greater than 0.7 achieve superior separation between clusters, while maintaining data points close to their assigned cluster centroid [70]. Values between 0.5 and 0.7 are also acceptable, indicating that the data points are sufficiently close to their cluster centroid while still maintaining separation between other clusters.

The second criteria is the degree to which robots are evenly distributed into k clusters. An even distribution will provide precise insights into the traffic characteristics of robots by clear differentiation. In contrast, lumping a majority robots into few clusters will lead to general conclusions without any distinctive insights. To measure our second criterion we consider the size of each cluster. In a desirable distribution of robots into clusters, the variance in the size of the clusters must be low, signifying that the robots are not overly concentrated into

a single cluster.

We examined both the measures because a high value of $SC_{\hat{C}}$ does not imply that the cluster size variance will low. A superior choice for k, for example, may be one where its value of $SC_{\hat{c}}$ is within an acceptable range and its cluster size variance is smallest. Once the data are partitioned into k clusters, each cluster is given a unique label $C_{a,b,...}$, where each subscript is assigned an integer value according to the rank of the cluster's centroid position in nondecreasing order for each respective traffic feature. This cluster labeling allows the scheme to be easily expandable to consider any number of data features.

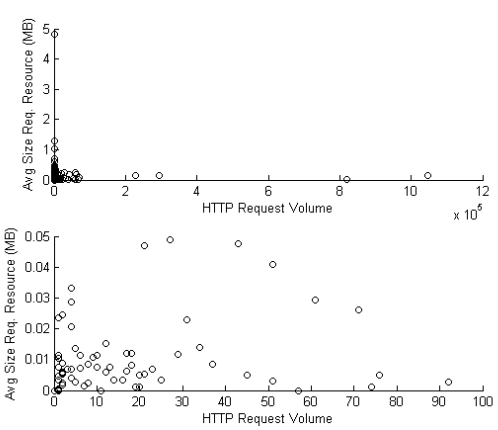
To apply this classification, we extracted three different properties of Web robot traffic: (i) volume of HTTP requests sent, (ii) volume of bandwidth consumed, and (iii) average size of resources requested. We analyzed the three metrics in a pairwise fashion over the entire set of robots to explore the correlations between them. Figures 3.7(a) through Figure 3.7(c) show the results of the pairwise analyses of these metrics. In each figure, the top plot includes all data points, while the bottom one focuses in on the most concentrated region to offer a better sense of the data distribution. Figure 3.7(c) shows a positive linear relationship between bandwidth consumed and volume of http requests, with the correlation coefficient measured at 0.804. This observation matches with previous results suggesting a strong linear correlation between request volume and bandwidth consumption for all server traffic [77]. On the contrary, the average size of requested resources

exhibits no observable relationship with both the request volume and bandwidth consumption (Figures 3.7(a) and (b)), with correlations of 0.035 and -0.004 respectively. These observations thus dispute the belief that a robot, which on average requests very large resources, will also consume a considerable bandwidth or will send a large number of http requests.

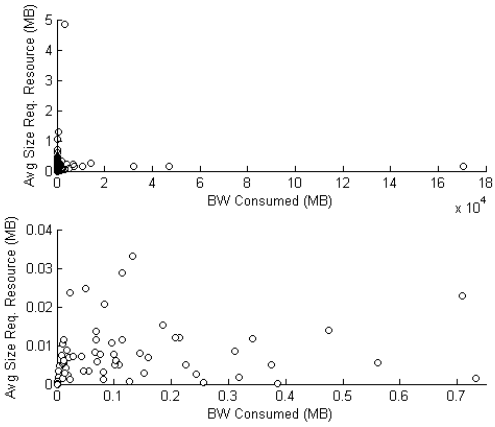
The top plots in all the figures indicate that some robots place disproportionate strain on the Web server. Although it is common to filter such outliers before applying clustering, we chose to include them because it is important to understand the traffic from these robots that disproportionately consume server resources from the point of view of server preparation.

Cluster analysis

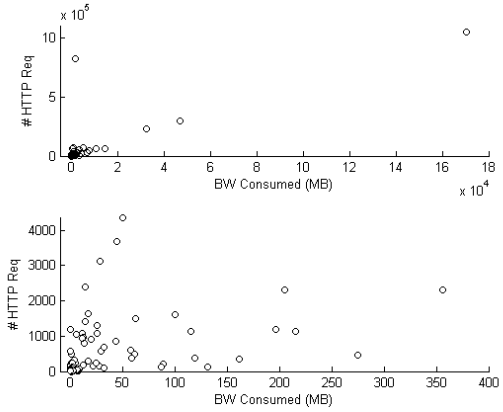
We performed K-means clustering with the three metrics for each of the 169 robots. We chose these three metrics to illustrate the feasibility of using clustering to partition Web robots; in practice any number of additional traffic metrics could be used to generate a higher-dimensional clustering. The clustering algorithm was implemented in MATLAB, and verified using several manually-generated test sets that contained clear groupings of the data points. In this section, we first discuss our analysis to select the appropriate number of clusters. Subsequently, we comment on the quality and characteristics of the clusters and the important insights they provide into robot traffic.



(a) Avg. Request Size



(b) Avg. Bandwidth



(c) Request Volume

Fig. 3.7: Pair-wise comparison of robot traffic characteristics

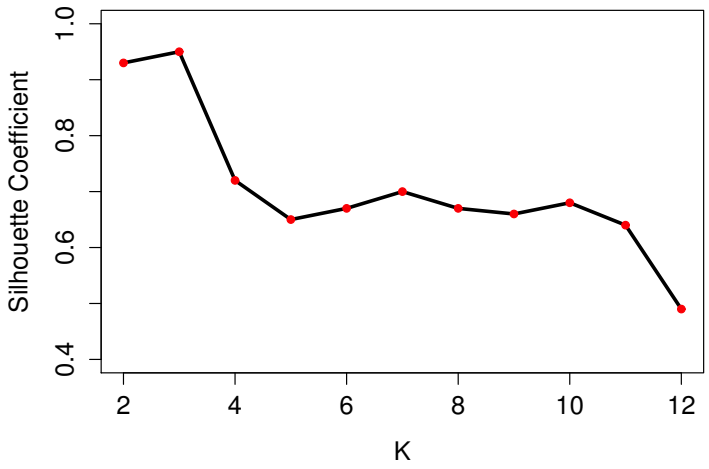


Fig. 3.8: Silhouette coefficients for each k-clustering

To select an appropriate number of clusters that maximizes the silhouette coefficient and minimizes the variance in cluster size, we performed K-means clustering with randomly selected initial centroids for k ranging from 2 to 12. We limited the maximum number of clusters to 12 due to the small number of robots.

Figure 3.8 plots the value of $SC_{\hat{C}}$ as a function of k. While $k = 2, 3$, and 4 show very high values of $SC_{\hat{C}}$, using so few clusters would offer little insights since this would not appropriately classify the outliers across any metric into its own group. A noticeable dip in the measure is seen when $k = 5$, followed by a steady increase until another peak at $k = 7$ where $SC_{\hat{C}} = 0.7038$. For $7 \leq k \leq 11$, the levels of the silhouette coefficient indicate a good tradeoff between inter and intra-cluster distances.

Figure 3.9 charts the variance in cluster size for the same range of k. When $k = 12$ the variance in size of each cluster is smallest, however, the respective

value of $SC_{\hat{C}}$ drops significantly. For $7 \leq k \leq 11$, the variance in cluster size is small and does not drop significantly as k increases. Recognizing a peak in the value of $SC_{\hat{C}}$ and relatively low variance for $k = 10$, we choose to partition these robots into 10 clusters.

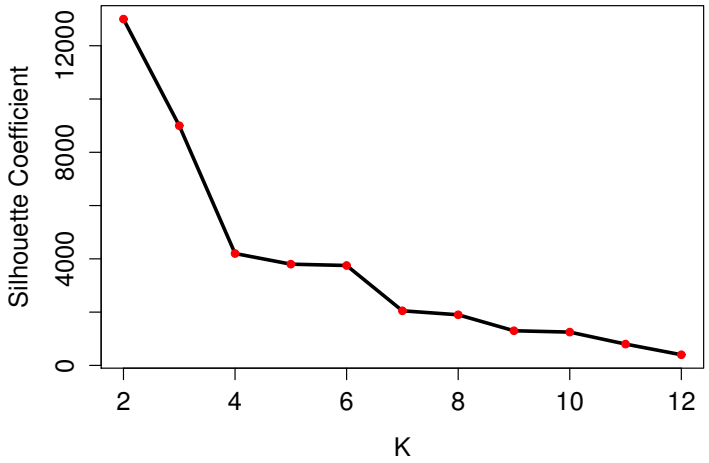


Fig. 3.9: Variance of size of each cluster for each k-clustering

Cluster Characteristics

Table 3.7 shows the average values of each metric or the coordinates of the centroid for each cluster. The clusters are assigned a label $C_{a,b,c}$ where a, b and c represent the cluster rank based on request volume, bandwidth consumption, and average size of requested resource respectively. Figure 3.10 presents a three-dimensional plot of the positions of cluster centroids, with a log scale for request volume and bandwidth, and a linear scale for the average requested resource size. The figure shows that the centroids are positioned along the request volume and bandwidth

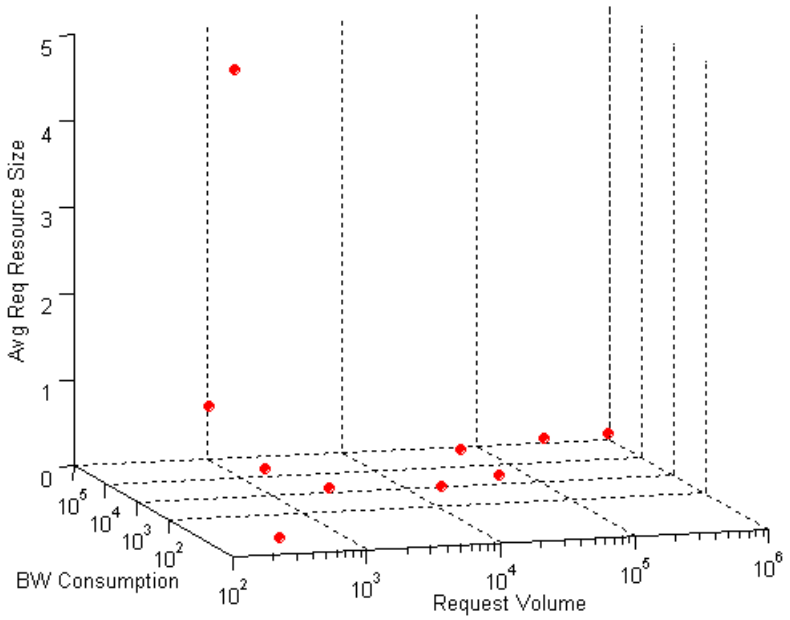


Fig. 3.10: Centroid positions for each cluster

axis according to the positive linear correlation observed between these metrics. The centroid positions along the average requested resource size axis, however, are concentrated because robots tend to request very small resources on average [30]. This is especially true for this academic Web server, which is likely to host a large collection of small files.

Table 3.7 also defines size and the boundaries for each cluster across the three metrics. The table reveals that over 63% robots fall into cluster $C_{2,1,1}$, whose label suggests that this group of robots request a relatively small volume of http requests, consume little bandwidth and request the smallest resources on average. The membership of this cluster is significantly high due to the presence of outliers, which are forced into their own cluster (for example, $C_{4,7,10}$ and $C_{1,3,9}$).

Because these outliers cannot be ignored, we can accommodate them by refining very large partitions through repeating the clustering only over robots in these partitions. This will produce a hierarchical structure of clusters where the highest-level ones deliver a broad classification of Web robots while lower-level clusters refine a broad class into a series of more specific ones. For example, Table 3.7 suggests that robots in $C_{2,1,1}$ exert low demands on the server. Furthermore, this large $C_{2,1,1}$ cluster also contains robots that do not retrieve any resources. Thus, it may be desirable to partition this cluster further to isolate such "no-demand" robots into their own class. Such refinement of clusters can classify robots at any desired level of granularity.

Since robots in this cluster consume relatively fewer resources, they most likely reflect traffic that does not impose significant strain on the server. The difference in the bounds along each metric is also small, which also indicates that robots in this cluster are heavily concentrated. By comparison, clusters of high-demand robots such as $C_{7,8,7}$, $C_{9,9,5}$, and $C_{10,10,4}$ are very wide and have few members. The few robots in this cluster show extraordinary characteristics, and hence, should be examined more closely to determine if their purpose is in the best interests of the Web server. If the investigation reveals that these robots are from commercial services that provide no benefit to UConn SoE for example, they should be blocked from access.

	Req. Volume			Bytes Transferred (MB)			Avg. Req. Size (MB)		
	min	max	avg	min	max	avg	min	max	avg
$C_{9,9,5}$	2.3E5	2.9E5	2.6E5	3.2E4	4.7E4	3.9E4	.142	.160	.151
$C_{5,4,6}$	2	15,559	2,103	.301	2,043	335.9	.088	.283	.154
$C_{3,2,8}$	1	5,329	634.6	.345	1,794	234	.307	.708	.447
$C_{2,1,1}$	1	4,351	339.8	0	99.7	5.91	0	.072	.014
$C_{10,10,4}$	8.2E5	1.1E6	9.3E5	1,534	1.7E5	8.6E4	.002	.163	.082
$C_{7,8,7}$	2.0E4	6.2E4	4.3E4	4,341	4.2E4	8,681	.171	.243	.206
$C_{6,5,3}$	8,347	3.2E4	1.8E4	.386	2,461	818.8	4.3E-5	.091	.036
$C_{4,7,10}$	717	717	717	3,466	3,466	3,466	4.83	4.83	4.83
$C_{8,6,2}$	3.9E4	7.0E4	5.8E4	182	4,928	1,956	.003	.071	.032
$C_{1,3,9}$	126	402	264	131.2	523.4	327.3	1.04	1.30	1.17

Table 3.7: Robot cluster statistics

3.5 Related Research

A number of efforts have studied the traffic characteristics of Web robots with an eye towards detecting such robots. Stassopoulou *et. al.* [108] employ a detection framework based on a Bayesian network, while Tan *et. al.* [112] perform detection based on the navigational patterns of Web robots. Through a more extensive study of robot traffic, focusing on crawlers that belong to five well-known search engines, Dikaiakos *et al.* [30] gain insights into their crawler behavior as a means for separating human users from robots in access logs.

The above efforts consider aggregate properties of robot traffic. In contrast, the research described in this paper applies data clustering to classify Web robots to gain a more detailed understanding of their specific traffic patterns. This exercise is necessary because modern sophisticated Web robots exhibit a wide

variety of functionality and visiting intentions, leading to a significant disparity in their crawling behaviors and demands [31]. A detailed study can form the basis of a scheme to detect and block ill-behaved robots. It can also lead to analytical models of robot workloads, which could be used to assess server performance.

3.6 Chapter Summary

This chapter presented a novel classification framework to understand and analyze the behavior of Web robots along multiple, orthogonal perspectives. The framework allows us to assign a multi-dimensional classification to a robot based on their functionality, the types of resources they prefer, and on the characteristics of their workload on a Web server. Using robots' classifications as a basis, we made the following observations about robot traffic:

- Over 23% of robots have unknown or undocumented functionality.
- 40% of HTTP requests from robots are from indexers.
- The average size of resources requested by indexing robots is not accurately estimated by the average size of HTML pages.
- 50% of robots favored *web* resources, while the robots from other resource classes do not favor *web* resources strongly.
- A robot that does not heavily favor *web* resources may have some pre-existing knowledge about the structure of the site before starting its crawl.
- Robots belonging to the *img* class overwhelmingly prefer *img* resources.

- The expected favoritism index of a robot's favorite resource type is nearly similar for *web*, *txt*, and *doc* robots.
- Robots which prefer *oth* resources do so with a higher average degree than any other resource class.
- Robots are best partitioned into 10 different characteristic classes. In other words, robots impose a wide range of demands on a Web server.
- Over 63% of robots request a small voume of resources, consume little bandwidth, and request small resources on average.
- 10% of robots visiting the UConn SoE Web server fall into clusters that represent disproportionately intense demands on a Web server.

Chapter 4

Workload Analysis

The behavioral, statistical, and workload properties of human-induced traffic on the Web are now well understand [7,57,77,26,6,54,127]. This understanding has led to the development of essential Web server optimizations [81], traffic generators [22,118,14], and models that forecast the workload of traffic on a Web server [61]. Since the present levels of robot traffic on the Internet is now substantial [109,60,30], and since our classification scheme revealed how robots implement advanced functionality [5,97] and exhibit widely varying behaviors [31,59,33,66], it is now important that we establish a similarly deep and thorough understanding of their traffic as well. It is unlikely that our understanding of human traffic will transcend to Web robots because the autonomous way in which Web robots send requests to retrieve information off a site stands in sharp contrast to the way humans browse a Web site. For example, a robot may send requests at a constant request rate to retrieve all the resources at a site, and then performs processing to extract knowledge after the fact. Human-induced traffic, however, is much more deliberate. A human visits a site with the goal of finding specific

knowledge, and analyzes the structure and data on a page to find the information desired. Furthermore, the traffic properties of humans are induced by the characteristics of a Web browser. These Web browsers quickly send requests for all embedded resources on an html page, causing human traffic to be bursty. Robots, however, may send requests at any time and at any rate. Consequently, due to the differences between humans and robots, we hypothesize that there must exist contrasts and distinctions in the statistical characteristics of human and robot-induced requests.

In this chapter, we throughly examine whether or not Web robot traffic exhibits two important statistical characteristics: (i) power-tails in their response sizes, interarrival times, session request volume, and intersession times; and (ii) long-range dependence in the arrival process of their traffic. These two characteristics are investigated in robot traffic across Web servers from three separate domains, namely, academic, research, and e-commerce, to promote the breadth of conclusions. In our search for power-tails, we use a unique multi-faceted analysis which suggests that only interarrival and inter-session Web robot request times are power-tailed whereas request volume and response sizes are not power-tailed. Furthermore, we find that robot traffic does exhibit long-range dependence in their arrival process, similar to humans. Our discussion following this analysis explains how the behavior of Web robots may give rise to non power-tailed trends, and discusses the implications of our findings on current policies and optimizations

employed by Web servers expecting power-tailed traffic. We also present how our analysis suggests that, unlike human traffic, the long-range dependence of robot traffic depends on the inter-arrival time distribution of their requests[1].

This chapter is organized as follows. Section 4.1 motivates the selection of the three Web servers through a preliminary analysis of their logs. Section 4.2 gives a brief primer on power-tailed distributions. Section 4.3 searches for power-tails in Web robot traffic through our unique, multi-faceted analysis. Section 4.4 analysis long-range dependence in the arrival process of Web robots. We review the related work in Section 4.5 and conclude the chapter in Section 4.6.

4.1 Preliminary Analysis

We analyze robot traffic from Web servers across three different domains over the same outage-free, seven-week period in 2009. In this section, we discuss how the domains represented by these three servers are diverse because of their services, types of Web applications, and the traffic trends observed on them. Examining Web robot traffic across these three assorted servers will thus provide a perspective that cannot be obtained through the analysis on a single server.

- (**Academic**) UConn SoE Web server. This server hosts an informational Web site that provides information about the school, the departments and faculty. The Web server also hosts all faculty, laboratory, course, and student

[1] Portions of this chapter were previously written and published by the author in [35,37,38].

Web pages.

- (**E-commerce**) UConn Co-op Web server. This server hosts an e-commerce store to purchase UConn branded merchandise and books. The store is powered by a Web application that allows users to browse for products without logging in. After being authenticated, the users can also purchase the merchandise.
- (**Research**) Roper Center for Public Opinion Research Web server. This server hosts an online database of public opinion polling questions and responses. A small amount of data can be accessed on informational pages through the site without registration, but the majority of polls, responses, and datasets can only be accessed through a Web application that requires registration. Users are granted automatic access if their ip address lies in a white-list of pre-authenticated addresses. This gives robots an opportunity to collect data that is only accessible through the Web application.

The primary purpose of the SoE server is to provide information; the few pages that require authentication are included for administrative tasks and are hidden from casual visitors. The primary purpose of the Co-op server is to facilitate online commerce and hence it hosts a Web application that supports functions such as browsing catalogs, adding items to a cart, and registering personal information in a secure database. The e-commerce server does not host any pages that can be considered as providing information in the same vein as the SoE server.

Finally, the Roper Center server combines the functions of both an academic and e-commerce server, because it hosts informational pages as well as an online Web application to acquire public opinion research data.

The logs on each server contain an entry for every http request that it receives. We pre-processed these entries to extract the requested resource, its size, the time of the request, the http response code, and the sender's user-agent field. From this pre-processed data, we obtained a sample of Web robot requests by comparing the user-agent field against a database of regular expressions representing well-known Web robots. The regular expressions in this database were collected from AWStats, a popular open source log analysis tool [9]. Although the previous chapter proposed an advanced approach for extracting robot traffic from a Web log [36,34,51,108], this simple approach is sufficient to obtain a large enough sample of Web robots to allow for statistical examination of heavy-tailed trends and long range dependence.

The preliminary summary of robot activity on each Web server is shown in Table 4.1, illustrating the large number of robot requests (at least 80,000 per server) collected during the seven week period across all three Web servers leaves us with a sufficient amount of data to perform a meaningful statistical analysis. We further explore the noteworthy distinctions in the profiles of robot traffic faced by each server next.

Characteristic	Web Server	Aggregate	Human Traffic	Robot Traffic (lower bound)
Total Requests	Coop	4,586,201	4,439,685 (96.8%)	146,516 (3.2%)
	Roper Center	1,400,073	1,317,175 (93.8%)	82,898 (6.2%)
	SoE	1,087,826	947,524 (87.1%)	140,302 (12.9%)
Avg. Req/day	Coop	95,545	92,493	3,052
	Roper Center	29,369	27,912	1,457
	SoE	22,663	19,740	2,923
Bandwidth (GB)	Coop	24.22	20.65 (85.3%)	3.57 (14.7%)
	Roper Center	15.32	12.27 (80.1%)	3.06 (19.9%)
	SoE	81.84	66.30 (81.0%)	15.55 (19.0%)
Avg. GB/day	Coop	0.50	0.43	0.07
	Roper Center	0.32	0.26	0.06
	SoE	1.71	1.38	0.33
Unique Requests	Coop	5,014	4,980 (99.3%)	2,555 (51.0%)
	Roper Center	69,213	64,396 (93%)	5,744 (8.3%)
	SoE	74,061	34,516 (46.6%)	58,115 (78.5%)

Table 4.1: Summary statistics of Web logs

4.1.1 Request volume

The proportion of requests from Web robots is the largest on the SoE server. We note that volumes reported in Table 4.1 are a lower bound on the actual volume of robot traffic, which may be much higher. The proportion of requests to the SoE Web server is twice as large as the proportion to the Roper server, which in itself is twice as large as the proportion to the Co-op server. The SoE server may be faced with the highest proportion of robot requests because robots visit this site frequently to stay abreast of the information updates. The Roper server sees an intermediate proportion of robot requests because it still serves some information, however, this information is updated only intermittently and remains fairly static. This information is also highly specialized compared to

the information available throughout the SoE Web server, which drives down the number of robots interested in crawling the server. Only the information avaialble through registration is updated frequently, but so few robots may have access to this that they do not appreciably increase the total number of requests sent. The very small proportion of Web traffic to the Coop site is due to its tremendous popularity by human visitors that made over 4 million requests during the seven week period. Besides traditional search engine indexers and Website scrapers, shopbots also visit the catalog to collect pricing data, or to browse the online catalog to collect metadata about products for price comparison Web sites and search engines.

4.1.2 Bandwidth

Nearly one-fifth of all data transferred to clients are sent to Web robots on the SoE and Roper Center servers, Furthermore, the total bandwidth consumed by robots on the SoE server is five times higher than the Roper Center, which may be due to the many different kinds of files the SoE Web server hosts, from images, videos, and faculty research papers, to massive datasets and compressed executables. In contrast, the Roper Center primarily has smaller html pages, images, and RSS feeds accessible throughout their Web site. The Coop has the lowest proportion of data sent to robots, due to its higher popularity. However, the total size of resources requested to robots fall between the Roper Center and the SoE Web

servers. The Coop serves more images, text, and embedded html files than the Roper Center to support its Web application, but the variations in file types and sizes are much smaller.

4.1.3 Unique requests

The total number of unique requests for distinct resources is a reflection of the accessibility of the data available on each Web server. Nearly all files hosted on the SoE Web server is available without the need to perform any authentication. Thus, robots were able to harvest nearly 80% of all files that are available on the server. The Coop e-commerce Web application is more restrictive. Robots can browse the entire site catalog, but is unable to reach shopping cart, registration, payment, and shipping pages without first logging into the site, which constitutes approximately half of all hosted files. Finally, almost all of the available data on the Roper Center Web server is located within its hosted Web application that requires an account to access.

Our preliminary analysis suggests that all three Web servers, representing different domains of the Internet and hosting different kinds of Web sites and applications, are faced with traffic that has distinct patterns. By analyzing robot traffic across these diverse Web servers, we will be able to identify statistical characteristics of their traffic which is invariant to its domain, to the services it provides, and to

the profile of traffic that they face.

4.2 Power-tailed distributions: An Overview

A distribution $F(x)$ is said to be *heavy-tailed* its reliability function $R(x) = 1 - F(x)$ drops at a rate that is slower than exponential [80], that is,

$$\lim_{x \to \infty} e^{sx} R(x) = \infty$$

for all $s > 0$. *Power-tailed* distributions are a strict subset of the set of heavy tailed distributions [80]. where the reliability function also satisfies the property:

$$R(x) \sim cx^{-\alpha}$$

where c is a constant and α is the scaling parameter, defining the rate at which the reliability function goes to zero. As α decreases, an increasingly larger portion of the probability mass is present in the right tail. While it still holds that $\lim_{x \to \infty} Pr[X > x] = 0$, the probability of witnessing extremely large values does not drop at least exponentially fast in contrast to more common distributions such as the Normal, the Poisson, and the Exponential [80]. We note that many legacy Web traffic studies use the term heavy-tail when they actually are referring to power-tailed behavior [27,40,6,30,77].

The defining characteristic of power-tailed distributions is that they exhibit infinite moments. This can be seen by noting that the probability density function

corresponding to $F(x)$ is

$$f(x) = -\frac{dR(x)}{dx} = \frac{c\alpha}{x^{\alpha+1}}$$

so its moments are given by

$$\mathbb{E}[X^l] = \int_0^\infty x^l \frac{c\alpha}{x^{\alpha+1}} dx$$

which when $l > \alpha$ evaluates to

$$\mathbb{E}[X^l] = A(z) + c\alpha \int_z^\infty x^{l-\alpha-1} dx = \infty$$

where z is the value of x where the power-tailed behavior begins, and $A(z)$ is the value of the integral from 0 to z. For example, if $\alpha < 2$, $\mathbb{E}[X^2] = \infty$, so X has an infinite variance. This indicates that no matter how long we sample X, there will always be an event larger than what our data captures. The large events, however, are at least balanced by all the small events between the large ones (in the sense that from a sample of n data points, the sample mean $\bar{x} \to \mathbb{E}[X]$ as $n \to \infty$) . If $\alpha < 1$ however, $\mathbb{E}[X] = \infty$, so the sample mean $\bar{x}$ diverges as $n \to \infty$. Thus, empirical measurements of a power-tailed phenomenon where $\alpha < 2$, provides limited insights because many sample statistics do not converge.

Power-tailed characteristics in the size or volume of resource requests have critical implications for computing systems that serve a large number of requests. Such systems must be specifically designed to handle the occasional extreme heavy load that will inevitably arise to maintain high availability and performance [8].

Thus, researchers have devoted significant effort to check for power-tailed trends in the traffic on the Web, which is a massively distributed system that is comprised of servers serving millions of requests.

4.2.1 Common power-tailed distributions

Our analysis of Web robot traffic looks for fits between empirical distributions and three theoretical distributions, namely the Pareto, Weibull, and Lognormal distributions. This section we briefly reviews these distributions and discusses their application in the analysis of Web traffic.

Pareto distribution

The Pareto distribution is power-tailed over its entire range [80]. The probability density function takes a shape parameter α and location parameter k. It is given by:

$$pt(x) = \frac{\alpha\theta^{\alpha}}{x^{\alpha+1}}$$

for $\alpha, \theta > 0$ and $x \geq \theta$.

The Pareto distribution has been used to model the size of Web server responses to humans that are greater than a certain threshold θ. Such a model is supported due to the fact that the sizes of resources available on a Web server follow a Pareto distribution [27]. Probabilistic analysis that predict Pareto response sizes from a Web server offer further support [88].

Weibull distribution

The Weibull distribution has a pdf given by:

$$f(x,k,\theta) = \frac{k}{\theta}(\frac{x}{\theta})^{k-1}e^{-(x/\theta)^k}$$

where k is a scaling parameter [80]. When $k < 1$, the Weibull is power-tailed. The Weibull is traditionally used by reliability engineers to model the probabilities of failure. In the context of Web traffic, we can model the interarrival times of requests to a server according to the an analogy with reliability modeling: we can say that a request made to a Web server is a "failure" event. Like many reliability models, the time between requests (failures) is likely to be very small during an active session (burn-in period), but the longer it has been since a client sent a request (failure event), the less likely it is that we will observe a request (failure) in the future.

Lognormal distribution

If $Y = \ln X$ be a random variable that is normally distributed. we say that X follows a *lognormal distribution*, with probability density given by:

$$logn(x) = \frac{1}{\sqrt{2\pi}\sigma x}e^{-(\ln x-\mu)^2/2\sigma^2}$$

where μ and σ are the mean and standard deviation of the normal distribution that Y follows. Note that although the Lognormal distribution obeys the limit

definition of heavy-tailed distributions based on the limit definition, it is not power-tailed because all of its moments are finite.

4.2.2 Verifying power-tails

In many studies of physical and economic phenomena [11,25], the method used to identify a power tail consists of a visual inspection of $R(x)$ plotted on a log-log scale and identifying a linear trend with slope $-\alpha$. This linear trend emerges because $\log R(x) = \log c - \alpha \log x$, and its derivative with respect to $\log x$ is equal to $-\alpha$. Because the physical and economic data sets very strictly exhibit this straight line behavior with values of $\alpha < 2$ [11], it can be visually ascertained whether the data is power-tailed or not.

Web traffic data seldom exhibits the precise power-tailed behavior that many natural phenomena follow. This can lead to many difficulties in asserting that a data set follows a power-tailed distribution. For example, a data set that is Lognormally distributed with a very large variance will show a linear trend over a range of values in a plot of its reliability on the log-log scale. This can be seen by taking the natural log of $logn(x)$:

$$\ln logn(x) = -\ln x - \ln \sqrt{2\pi}\sigma - \frac{(\ln x - \mu)^2}{2\sigma^2}$$

When $\sigma^2 \to \infty$, the quadratic term drops to 0, leaving a linear function. As visual fits can be inconclusive, researchers have proposed a variety of techniques for verifying the existence of power tails in empirical data [123,119,40,39]. For

our study of robot traffic, in addition to a visual analysis of the best fitting power tailed distributions to empirical data, we consider two statistical tests and look for a consensus among these tests. These tests include the Hill estimator and Vuong's distribution comparison test.

Hill estimator

The Hill estimator [123] is commonly used to estimate α for a power-tailed distribution because its estimation procedure starts at the far right tail of an empirical data set and then works backwards, mitigating the impact that small values, which do not contribute to the power-tailed trend, have on the estimation. It is defined as follows: let $x_1, x_2, ..., x_n$ denote the data set generated by the random variable X and $x_{(1)} \geq x_{(2)} \geq ... \geq x_{(n)}$ be the order statistics of the data. Using $k < n$ of the order statistics, the Hill estimator of α is given by:

$$\hat{\alpha}_k = (\frac{1}{k}\sum_{i=1}^{k} \log \frac{x_{(i)}}{x_{(k+1)}})^{-1}$$

The change in the estimator $\hat{\alpha}_k$ over values of k is then plotted up to k_s, the order statistic that represents the start of the power tail. For small values of k the plot will be very noisy, but should gradually stabilize as k increases and a larger portion of the power tail is considered. If a power tail exists, the Hill estimator should stabilize to $0 \leq \alpha < 2$ as k approaches k_s on the Hill plot.

For our Web robot data sets, we construct plots of the Hill estimator to check whether α stabilizes. We consider all possible starting positions by varying

k across the entire range of the order statistics. In this plot, a power tail would be presented as a range of values for which α is stable, before varying more significantly as the smallest values in the data set are considered. The plots also include bands representing a 95% confidence interval for the estimates [20].

Distribution comparison

Comparing how well an empirical distribution fits against a theoretical one is a routine procedure when characterizing trends in a dataset. These distribution comparison methods are typically driven by a goodness-of-fit test where a distance from an empirical distribution to a theoretical one is computed and the significance of this distance is assessed by a test statistic. For example, the Kolmogorov-Smirnov (KS) test is based on the largest difference between the values of an empirical and theoretical distribution across their supports [78]. In the context of Web traffic, however, goodness-of-fit tests are very likely to reject the hypothesis that the data is power-tailed for two reasons. First, because Web data seldom precisely follows a theoretical distribution, the tail behavior of the data set will exhibit a high degree of variability. Second, a goodness-of-fit statistic can be distorted by the right tail of sampled power-tailed data, which we define as the m largest samples drawn out of n. This can be seen on a log-log scale of such power-tailed data, where the extreme right tail of the reliability function consists of a sparse number of points that can deviate from the linear trend [80].

To avoid the pitfalls of goodness-of-fit tests, we instead use a likelihood ratio test to statistically determine if the data follows a power-tailed distribution. We first compute the likelihood that the data follows either a Pareto or Weibull distributions, which are power-tailed, and test these fits against the non-power-tailed Lognormal distribution fit. Vuong's statistic [119] allows us to test whether one model has a significantly better fit to another model derived from a different probability distribution. It tests the hypothesis that both models equivalently describe the empirical data, against the alternative hypothesis that one of the models offers a superior fit. Let $f(X|\hat{\Theta})$ be one possible model of the data X fitted by parameters estimated through the maximum likelihood estimation (MLE) technique $\hat{\Theta}$, and $g(X|\hat{\Psi})$ be an alternative model of X fitted with the MLE parameters $\hat{\Psi}$. Further, define:

$$LR(\hat{\Theta}, \hat{\Psi}) = \sum_{i=1}^{n} \log \frac{f(x_i|\hat{\Theta})}{g(x_i|\hat{\Psi})}$$

as the log-likelihood ratio between the models f and g. Vuong showed that when f and g are not nested, under the null hypothesis $H_0 : f(X|\hat{\Theta}) = g(X|\hat{\Psi})$:

$$V_d = \frac{LR(\hat{\Theta}, \hat{\Psi})}{\sqrt{n}\hat{\sigma_n}} \sim \mathcal{N}(0, 1)$$

where:

$$\hat{\sigma_n^2} = \frac{1}{n}\sum_{i=1}^{n}(\log \frac{f(x_i|\hat{\Theta})}{g(x_i|\hat{\Psi})})^2 - (\frac{1}{n}\sum_{i=1}^{n}\log \frac{f(x_i|\hat{\Theta})}{g(x_i|\hat{\Psi})})^2$$

The test considers the alternative hypotheses $H_f : f$ better describes the data, and $H_g : g$ better describes the data. Under H_f, $V_d \xrightarrow{a.s.} \infty$, while under H_G, $V_d \xrightarrow{a.s.}$

$-\infty$. Thus, we only need to select a critical value c from the standard normal distribution corresponding to the level of significance desired. By symmetry of the standard normal, we can perform the following two-sided significance test: if $V_d > c$, reject H_0 in favor of H_f, or if $V_d < -c$, reject H_0 in favor of H_g. Otherwise, do not reject H_0. We test V_d at the 95% confidence level, corresponding to $c = 1.960$.

Because the Weibull and Pareto exhibit a distinctly different behavior in its tail compared to the Lognormal distribution, we plot V_d for subsets of the order statistics, from $x_{(1)}$ to $x_{(m)}$, while varying m. In all of our tests, we select f as the Lognormal and g as either the power-tailed Pareto or Weibull distributions. From this construction, we would expect that $V_d > c$ when empirical data does not favor one of the power-tailed distributions. If the data is power-tailed, however, V_d should be negative and then exhibit a decreasing trend as we exclude a more significant proportion of the largest points in the data set. It must be emphasized that the results of the Vuong test do not imply that the data truly follows a Lognormal, Pareto, or a Weibull distribution. Rather, we interpret the results as a suggestion about whether the data more strongly exhibits power- or non-power-tailed characteristics. This suggestion can then be considered along with information from the reliability function plot and the Hill estimator to infer whether the data exhibits a power-tailed trend.

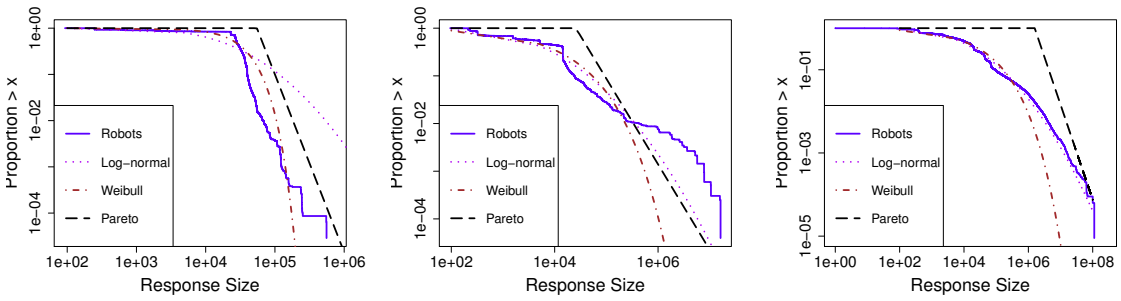
In our study, we use both the Hill estimator and Vuong's test statistic to analyze whether a power tail is present. The Hill estimator provides a precise view of the power-tailed nature of the distribution, in that we are able to identify how α, the main parameter that characterizes power-tailed behavior, changes as we consider more points from the right-tail of the distribution. The information provided by Vuong's test offers a more global view in that it considers the entire body of the distribution and only a portion of its right-tail These macro and micro level examinations of power tailed behavior help to improve the fidelity of our conclusions.

4.3 Power-tails in Web Robot Traffic

Because many properties of human Web traffic exhibit power-tailed trends [26,6,54], a natural question is whether robot traffic also exhibits such trends. If robot traffic does not exhibit power-tailed characteristics like humans, it would suggest the need to re-evaluate the practical implications of power-tailed behavior on Web server optimizations. In this section, we thoroughly examine whether the traffic induced by Web robots exhibits power-tailed characteristics similar to human-induced traffic. We compare human and robot traffic using two session-level (response size and number of requests per session) and temporal-level (interarrival and intersession times) metrics that critically influence the performance of Web servers [8,27,14,13]. Our multi-faceted analysis approach takes readings

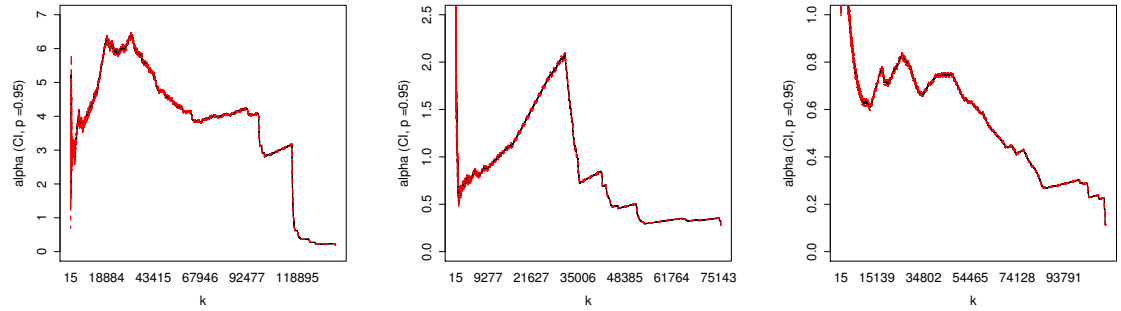
from several different that helps us convincingly conclude whether these features of robot traffic are power-tailed. In this section, we first provide an overview of power-tailed data along with a discussion of the commonly used distributions for modeling power-tailed trends. We then introduce the statistical tests used our multi-faceted analysis.

This section examines whether or not power-tailed trends exist in robot traffic across each of the three servers presented in Section 4.1. To consider both session and temporal features, we further separated the requests into sessions by identifying time lapses of at least 30 minutes between consecutive requests from the same IP address and user-agent field [86]. We consider traffic characteristics representing request-level and session-level features. Features at both levels are considered because request-level features are related to the arrival process of requests, while session-level features are associated with the activity that occurs within a session. The emergence of power-tails at both levels have been studied extensively in the literature for human traffic and in a limited fashion for Web robots [30]. For each level, we choose two characteristics that are extensively relied on in modeling Web server performance [81] and for Web traffic generation [22,118,14]. At the request-level, we examine interarrival and intersession times, and at the session-level we consider response size and the volume of requests sent per session.



(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.1: Response sizes: distribution fits

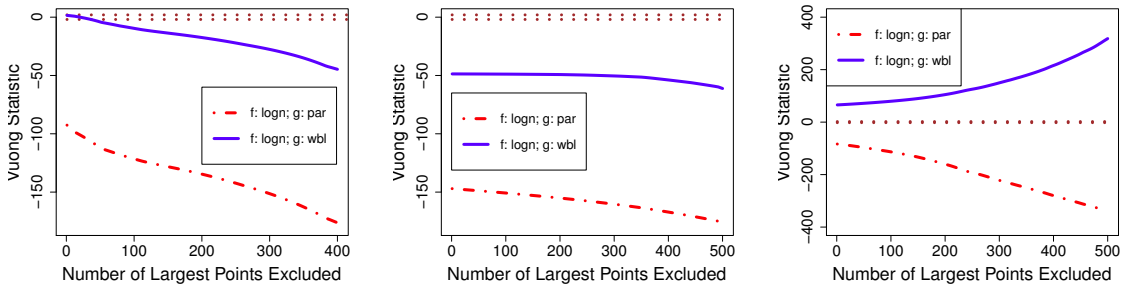


(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.2: Response sizes: Hill estimates

4.3.1 Response size

The response size is defined as the size of a resource that was requested and does not include the size of headers, flags, checksums, or any other information sent within an http response packet. The distribution of response sizes for each of the three Web servers, along with the best fitting Lognormal, Weibull, and



(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.3: Response sizes: Vuong statistics

Pareto distributions, is given in Figure 4.1[2]. At a first glance both the Coop and SoE servers show a linear trend that is indicative of power-tailed behavior. On the Coop server, the linear drop begins suddenly at approximately 10KB, however the far right tail is distorted by the few requests that have very large response sizes. The response sizes on the Coop server appear to fit the Pareto distribution ($\alpha = 3.85$) well, except for values at the extreme right tail that causes θ (the value at which the distribution begins to be fitted to the data)

[2] To properly fit a Pareto distribution, θ must be estimated first, and then the the best fitting value of α only for points larger than θ should be used. Clauset *et al.* proposes that θ be selected so that the probability distribution of the data and of the best fitting Pareto distribution be as close as possible for all $x > \theta$ [23]. In this study, whenever we fit a Pareto distribution to empirical data, we do so only for values larger than $\hat{\theta}$, an estimate of θ that minimizes the Kolmogorov-Smirnov goodness-of-fit statistic between the CDF of the data and of the best fitting Pareto distribution for $x > \hat{\theta}$. Specifically, if $F(x)$ is the CDF of the data and $P(x)$ is a Pareto distribution fitted to the data, θ is given as:

$$\theta = \min_{\hat{\theta}} \max_{x>\hat{\theta}} |F(x) - P(x)|$$

While the Kolmogorov-Smirnov statistical test is extremely sensitive to small deviations in a power tail, the value of the test statistic will still be minimized for the best fitting theoretical model and is suitable to find a minimum θ.

to be inaccurate. Response sizes on the Roper Center, with its straight-lined left tail, convex body, and exponentially decreasing right tail, are dissimilar to the three fitted distributions. Finally, the SoE server distribution experiences an increasing rate of decay approaching its tail that matches the shape of the Lognormal distribution.

Figure 4.2 plots the Hill estimator of α. The x-axis corresponds to the number of largest order statistics used in the distribution and ranges and starts at $k = 15$. To make the computation of these hill plots feasible, we note that the estimate was not computed only for values of m where the value of the order statistic changed. This causes the Hill plots to appear "jaggy", particularly for large values of k. However, since we seek to merely find a a stable trend in the Hill estimator and not find a precise estimate of α, this effect does not impact our analysis. For response sizes on the Coop, the Hill plot (Figure 4.2a) is very noisy as it takes over 50,000 out of 146,516 order statistics to be considered before the estimate begins to stabilize at approximately $\alpha = 3.9$. While this estimate matches the MLE of the Pareto fit, the amount of noise across such a remarkable number of points indicates that if a power-tailed trend existed, it would be unstable over a very large range. This forces us to reassess whether the linear trend in Figure 4.1a is consistent enough to consider it as a power-tail. The Roper Center and SoE Hill plots do not identify a stable value for α, supporting the hypothesis that a power tail does not exist.

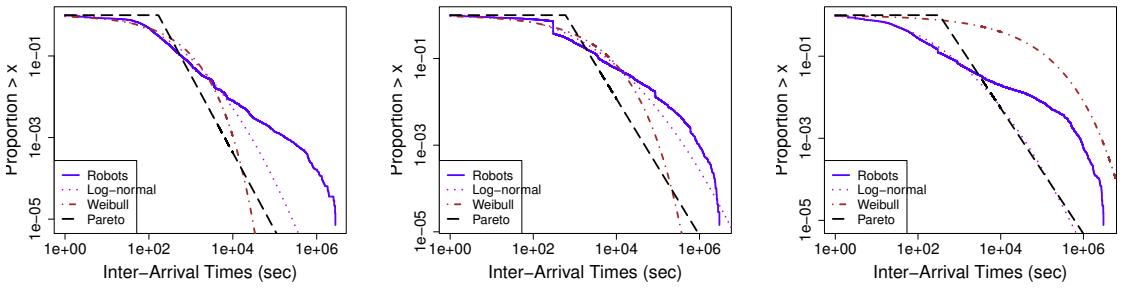
In Figure 4.3, Vuong's statistic is used to compare the Lognormal against the Pareto and Weibull fits. The straight dotted lines in the figures represent the threshold where we reject the hypothesis that both the Lognormal and one of the power-tailed distributions adequately describe the data in favor of the Lognormal ($V_d > 1.96$) or the power-tailed Pareto/Weibull ($V_d < -1.96$). We find that for the Coop and Roper Center servers, the Pareto and Weibull distribution offer a stronger fit to the data compared to the Lognormal. Considering the MLE distribution fits in Figures 4.1a and 4.1b, however, we see that this result may not be because the data tends to be power-tailed, but because the Lognormal distribution is a very poor fit to the data. On the SoE server, the Pareto distribution fits better than the Lognormal, but the Lognormal fits better than the Weibull distribution. The test indicates that the Pareto is a statistically better fit to the distribution compared to the Lognormal, even though the Lognormal fit closely follows the trend in Figure 4.1c and the Hill estimate (Figure 4.2c) never stabilizes to a consistent value of α.

To summarize, the three tests were unable to come to an agreement about whether or not response sizes are power-tailed across every Web server. The distribution fits and the Vuong test statistic suggest that robot traffic to the Coop Web server may have power-tailed response sizes, however the Hill plot cannot find a stable estimate of α until almost one third of all the points in the distribution are eliminated. Furthermore, Vuong's test suggest power-tailed response sizes

on the SoE Web server, despite an inadequate Pareto fit and unstable Hill plot. From this evidence, we cannot conclude that the response sizes of robot traffic is power-tailed.

We hypothesize that the reason robot traffic is not power-tailed is because the the aggregate response size distribution for Web robots may be unrelated to the distribution of resource sizes on a Web server, which is a direct reason for the emergence of power-tailed response sizes for human traffic [27]. This is supported by the following reasoning given by Crovella *et al.*: if all client-side caches used by Web browsers were considered as a collection, and that clients do not typically send requests for resources outside of their individual cache, then the combined resources stored in all client-side caches approximate all of the resources available on the Web server [27]. Assuming that all clients have a sufficiently large cache that is seldom emptied, the distribution of the sizes of all responses sent should approach the distribution of resources available on the server as more and more clients are considered. This is intuitive, since if a resource that was previously requested resides in the cache, the browser appends the *If-Modified-Since* header to the http packet, telling the server not to respond with the content if the file has not been modified since a certain time.

A similar argument may not hold for Web robots since it is not assured that all robots maintain a client-side cache of the requested resources. Furthermore, it is possible that different types of robots request different types of resources on



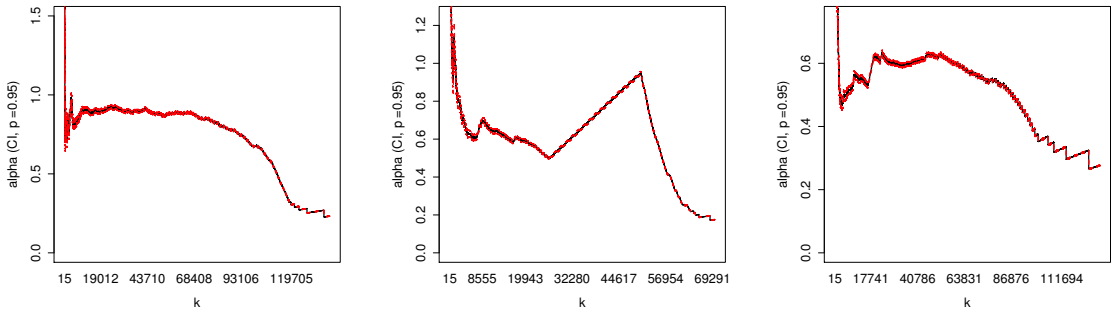
(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.4: Inter-arrival times: distribution fits

a site exclusively. For example, a search engine robot that seeks only academic papers may crawl a site and only send requests for document type resources, such as pdf, ps, and doc files. An e-mail harvester robot, on the other hand, may send requests for all types of resources except multimedia files to extract e-mail addresses.

4.3.2 Inter-arrival times

Inter-arrival times correspond to the time between requests originating from the same Web robot. These times were computed from the server logs by finding successive requests with identical user-agent fields and sender's IP addresses. The distributions of inter-arrival times plotted in Figure 4.4 all have linear bodies with tails that suddenly drop off at a position that corresponds to the length of the log analyzed. The limited length of the log truncates the inter-arrival times to never extend beyond seven weeks. We note, however, that this truncation phenomenon

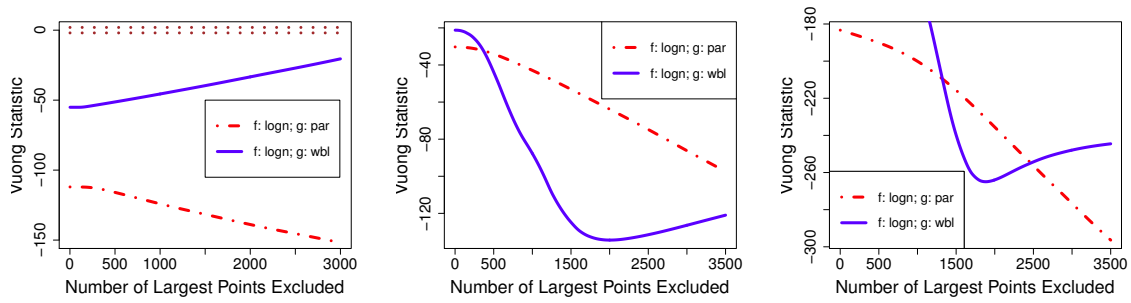


(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.5: Inter-arrival times: Hill estimates

would occur regardless of the length of the log considered in the analysis. This truncation causes points to accumulate in the far right tail, where all of the data points that are longer than 48 days in reality are measured as such. The best-fitting Lognormal distributions do not closely match the robot inter-arrival times on any server. Furthermore, the estimate of α for the Pareto fits across each server is inaccurate, but this may be be caused by the concentration of points due to the data truncation. It is thus difficult to assess the quality of the Pareto fits for this distribution or to perform any other kind of visual assessment of the fit.

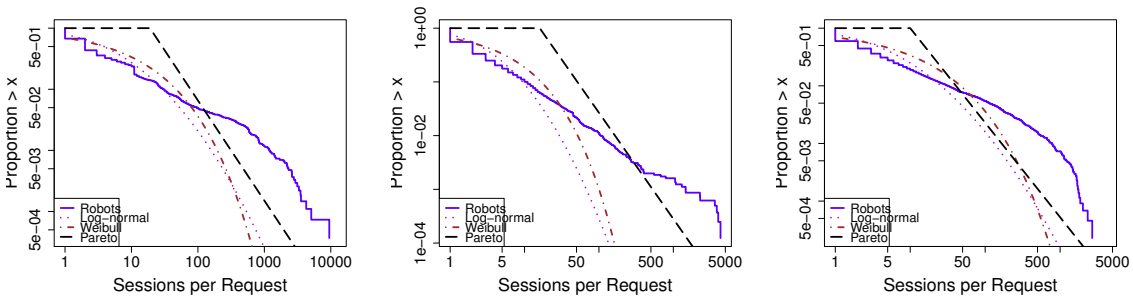
Despite the inconclusive distribution fits, can still rely on the Hill estimator and Vuong statistic to evaluate power-tailed behavior in truncated data sets. The Hill plot is effective with truncated data because the influence that the concentration of data at the right tail becomes mitigated as k increases. Vuong's test is also effective because the largest values in the data where points accumulate become



(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.6: Inter-arrival times: Vuong statistics

excluded. The Hill estimators for inter-arrival times are plotted in Figure 4.5. For the Coop Web server, the estimator stabilizes quickly to approximately $\alpha = 0.9$. The estimator for the Roper center stabilizes briefly before a sudden linear incline. This means that the distribution in Figure 4.4b has a power-tailed, linear shape only briefly near its truncated tail. The Hill plot for the SoE server is less stable than the plot for the Coop server, and features wide 95% confidence intervals. The wide confidence intervals can be accounted for by the small perturbations in the linear trend that run through the body of the distribution in Figure 4.4c. Across all three Web servers, The Vuong statistics plotted in Figure 4.6 very strongly reject the Lognormal fits for the power-tailed Weibull and Pareto fits across all of the servers. Given that the Hill plots are stable at least for a certain period, and that Vuong's statistic rejects the Lognormal strongly in favor of the Pareto and Weibull, we conclude that robot inter-arrival times exhibit power-tailed behavior.



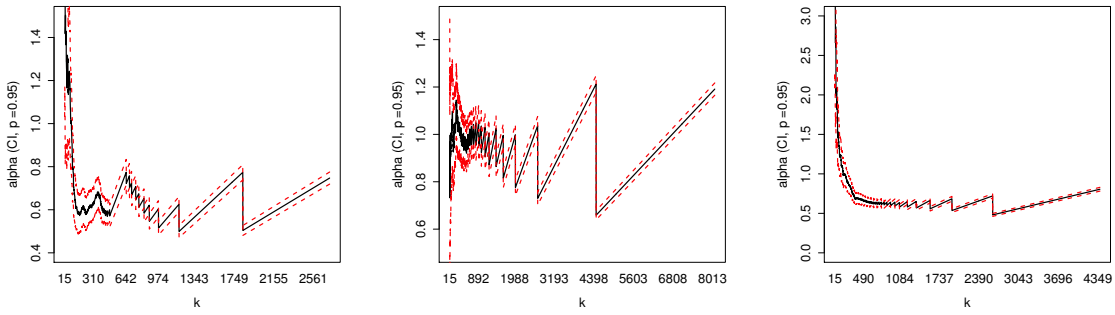
(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.7: Session request volume: distribution fits

4.3.3 Session request volume

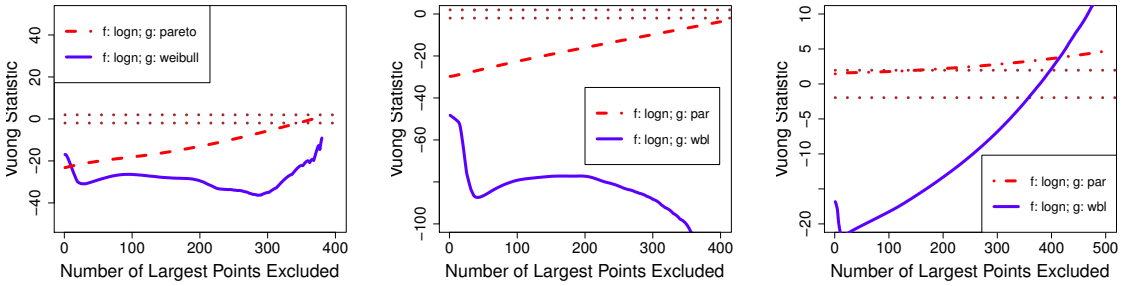
Next, we examine the number of requests that were sent per session by Web robots. Figure 4.7 presents this distribution across the three Web servers. The Roper Center exhibits a linear trend up until approximately 500 requests, where the slope suddenly changes and quickly drops. The Coop and SoE distributions show similar behavior, but the change in slope beyond 500 requests is more pronounced. Neither the Lognormal, Weibull, or Pareto distributions provide adequate fits to the data. Similar to the response size distributions, however, the poor Pareto fits may be influenced by the largest values in the extreme right tail.

Figure 4.8 Gives the Hill estimates for the three servers. The estimate for the Coop Web server fluctuates over a small range (between $\alpha = 0.55$ and 0.7), but has a wide confidence interval. The Hill estimate for the Roper Center varies over a much wider range of values, and becomes distorted as the number of order



(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.8: Session request volume: Hill estimates



(a) Coop **(b)** Roper Center **(c)** SoE

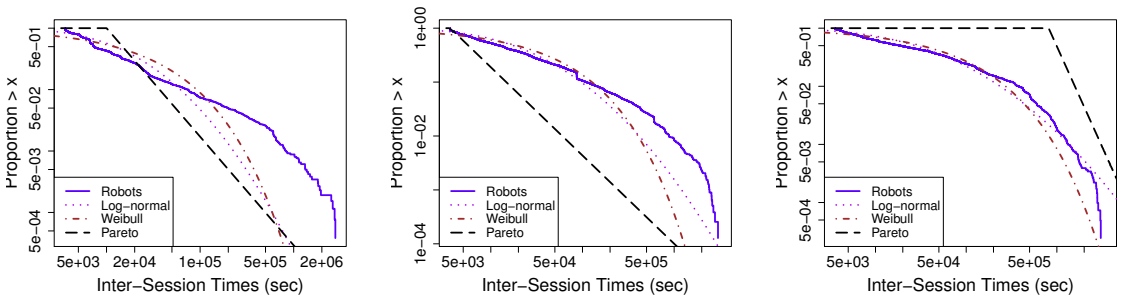
Fig. 4.9: Session request volume: Vuong statistics

statistics considered increases. For the SoE Web server, the hill estimate quickly stabilizes at approximately $\alpha = 0.75$, with a tighter confidence interval than the Coop. These observations hint at the presence of power-tailed behavior on the Coop and SoE Web servers. The plots of the Vuong statistic for these two servers in Figure 4.9 further suggest that a power tail is present. However, Vuong's test statistic increases as more points from the tail of the distribution are removed,

suggesting that the body of both distributions tend be better fitted by a Lognormal distribution than a power-tailed one.

The totality of our observations paint a conflicting, inconclusive picture about whether the volume of requests per session exhibit a power tail. Neither the Pareto, Weibull, or Lognormal offer a good visual fit to the dataset, the Hill estimator finds stability for the Coop and SoE, and the Vuong statistic suggests that the distributions have a tail that fits well to the Pareto or Weibull but a body that fits better to a Lognormal distribution. Given that no clear conclusions can be reached, we do not consider the distribution of the volume of requests per session to have a power-tail.

An intuitive reason why session request volume may not be power-tailed is because the number of requests per session is primarily related to the information sought during a visit to a Web site. While some human visitors send many requests per session as they continue to browse and collect new information, most visitors will briefly visit a site to collect specific information and then leave. These human visitors will all exhibit a similar navigational pattern and send a similar number of requests as they navigate to the same information before leaving. Furthermore, human users that make repeated trips for similar information will send even fewer requests as most resources may be available in their client-side cache. As a result, the number of requests in a session is likely to be similarly low for most sessions, but a few will exhibit extremely long sessions that contain many more requests.



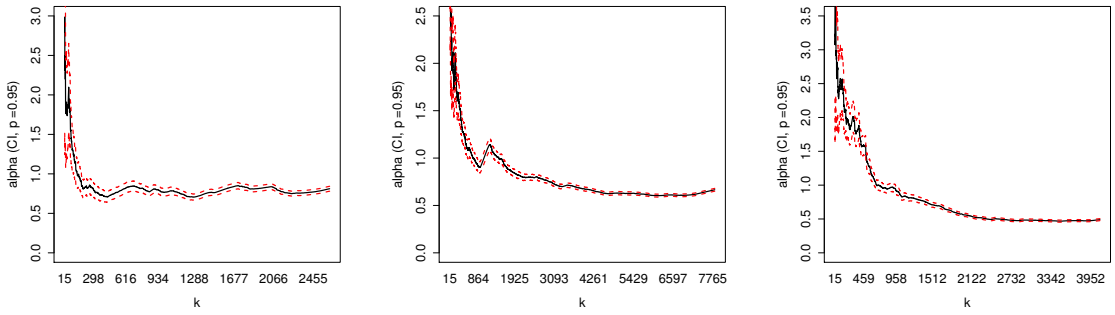
(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.10: Inter-session times: distribution fits

Web robots may contrast from this human behavior in two respects. First, it is unlikely that two different robots will traverse the same path and request the exact same resources per session, because of their diverse behavior and varying intentions [77]. As a result, the number of requests per session for Web robot traffic will be more diversified. Secondly, individual Web robots may send a fixed number of requests per session for all visits and perform the exact same crawl over and over again while not maintaining any client-side cache.

4.3.4 Inter-session times

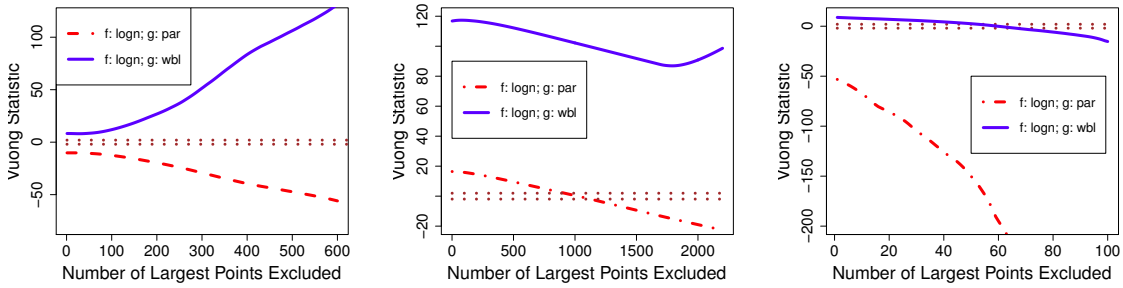
Finally, we examine the times between Web robot sessions. These distributions, shown in Figure 4.10, are artificially truncated like the interarrival time distributions. Prior to the truncation, we observe a linear trend in the data that has a consistent slope spanning over five orders of magnitude. This large span with a steady slope suggests that intersession times are power-tailed, but the truncation



(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.11: Inter-session times: Hill estimates

in the data prevents any conclusions from being drawn.



(a) Coop **(b)** Roper Center **(c)** SoE

Fig. 4.12: Inter-session times: Vuong statistics

The Hill plots presented in Figure 4.11 provide some evidence that interses-sion times exhibit power-tails. Across all Web servers, the Hill estimates quickly converge and remain steady around an α value with tight 95% confidence intervals. Unlike the Hill estimates for interarrival times, where the plot became distorted as an increasing amount of the distribution's left tail was considered, these plots

do not diverge away from the value of α it initially settles on. Thus, whereas interarrival times appear to be power-tailed only beyond a minimum data value, intersession times exhibit power-tailed behavior starting at the beginning of the distribution.

The Vuong statistics plotted in Figure 4.12 are in agreement with the Hill estimates and with the observations made about the shape of the intersession time distributions. We find that as the truncated tail of the distribution is eliminated, Vuong's statistic more strongly accepts the Pareto as being a superior fit versus the Lognormal distribution. On the Coop and SoE servers, we can even conclude that the Pareto fits better than the Lognormal without removing any points from the tail. For the Roper Center Web server, however, it takes the removal of approximately $1,250$ data points from the truncated tail before the Vuong statistic passes the threshold to accept the Pareto fit on the Roper Center server.

Given the agreement among our observations in the distribution shape, the Hill estimator, and Vuong's statistic, we conclude that intersession times for robot traffic are power-tailed. Our conclusion is consistent with the observation that the session-based time series of Web traffic is stationary [46]. If the process was not stationary, the power-tailed characteristics could have been the result of a moving average within the time series data.

We summarize the results of our comprehensive study in Table 4.2 where we list

Feature	Server	Dist. Fit	Hill Plot	Vuong	Conclusion
Response Size	Coop	Yes	No	Yes	**Not** Power-tailed
	Roper Center	No	No	Yes	
	SoE	No	No	Yes	
Interarrival Times	Coop	NC	Yes	Yes	Power-tailed
	Roper Center	NC	Yes	Yes	
	SoE	NC	Yes	Yes	
Session Request Volume	Coop	No	Yes	NC	**Not** Power-tailed
	Roper Center	No	No	No	
	SoE	No	Yes	NC	
Intersession Times	Coop	NC	Yes	Yes	Power-tailed
	Roper Center	NC	Yes	Yes	
	SoE	NC	Yes	Yes	

Table 4.2: Evidence of power-tailed behavior

whether or not the distribution fits, Hill estimate, and Vuong statistic suggests the presence of power-tailed behavior. We use "NC" in to denote "no conclusion" when a test's results were inconclusive. The last column lists our conclusion about whether or not a traffic characteristic is power-tailed, where we conclude that a characteristic has a power-tail if across all three servers, no test suggested that the trend was not power-tailed.

4.4 Long-Range Dependence

Long Range Dependence (LRD) is a property of time series processes that are self-similar, that is, where the correlation between measurements increasingly farther away does not appreciably diminish. Because of these very long-range correlations, a plot of the number of events per unit time does not smooth out as measurements become coarser. LRD, along with heavy- and power-tailed trends, is among the most important statistical properties of traffic for designing high-performance Web servers [27,52]. For example, exploiting LRD in Web traffic has led to the development of performance models for Web servers that are more accurate compare to queueing analysis [56]. Web servers have also been shown to be more energy-efficient so long as the incoming traffic demonstrates self-similar qualities [28].

It is not obvious that robot traffic exhibits LRD simply because human traffic has this property. This is because although robots send requests at a constant rate to retrieve many different, possibly unrelated resources, humans deliberately visit a site to retrieve specific information and will request resources according to the behavior of a Web browser. This behavioral difference suggests that robots may sport an arrival process that is fundamentally different from humans. Therefore, in this this section, we examine the arrival process of Web robot requests to determine if they also exhibit LRD.

4.4.1 Measuring long range dependence

Given a stochastic process $X = \{X_t | t = 1, 2, ...\}$ where each X_t is sampled over the same time interval, define the m-aggregated time series $X^{(m)} = \{X_t^{(m)} | t = 1, 2, ...\}$ by summing the values of the original process over non-overlapping consecutive ranges of size m (m is also referred to as the lag of the time series). Then, X is m-self-similar with parameter H if

$$X_t = m^{1-H} X_t^{(m)}$$

for all t, that is, if the m-aggregated time series is proportional to the original process at a finer scale. Furthermore, X is long-range dependent if both processes have the same asymptotic variance and autocorrelation as $m \to \infty$. If the process is LRD, as the lag $m \to \infty$ the autocorrelation function will behave as

$$\rho(m) = \frac{\mathbb{E}[(X_t - \mu)(X_{t+m} - \mu)]}{\sigma^2} \sim cm^{-\alpha}$$

where μ is the mean value of the process, σ^2 is its variance, and $0 < \alpha < 1$. The impact of this power-law decay can be realized by summing across all the lag m autocorrelations. When the process is not LRD, $\sum_{m=0}^{\infty} \rho(m) < \infty$, thus beyond some lag m the autocorrelation will drop to zero as future measurements become independent of the past ones taken from longer than some lag period ago [24]. When the process is LRD, however, the sum is infinite. This means that in a LRD process the autocorrelation depends on an infinite number of previous measurements.

A statistical approach to determine if a process is LRD is to estimate its Hurst parameter [24] which is defined as $H = (1-\alpha)/2$. Because of the restrictions on α, H must fall between 0.5 and 1 if and only if the process is LRD.

4.4.2 Hurst parameter estimation

Estimating the hurst parameter H from a dataset is difficult. A wide variety of estimation techniques have been proposed and analyzed, but they have been shown to give inconsistent results even when they are evaluated using artificially generated data [24]. Different estimators may be biased for large or small values of H, and can perform poorly depending on the periodicity, non-stationarity, and noise of the time series. LRD is unlikely to exist, however, if several estimators cannot provide a sufficient estimate of H. Thus, a common approach is to employ a suite of tests and then verify that the estimate of H returned by each are consistent. The estimators we choose are the R/S plot, the periodogram, wavelet analysis, and the whittle estimator. These tests were selected because they are commonly employed together to identify long-range dependence.

R/S Plot

The R/S statistic considers the expected value of the time series rescaled as follows. For the m^{th} data point in a time series $X = \{X_i\}$, define $Z_m = \sum_{j=1}^{m} X_j$, $R(i) = \max(Z_1, ...Z_m) - \min(Z_1, ...Z_m)$, and $S(i)$ as standard deviation of the time series

values $X_1, ..., X_m$. The expected value of the ratio $R(m)/S(m)$ goes to

$$\mathbb{E}[R(m)/S(m)] \sim cm^H$$

as $m \to \infty$ with c as a constant. When we plot these ratios generated using the time series X on log-log scale, we will find that they exhibit a linear trend with slope H. Thus, if X is LRD the slope of this trend should fall between 0.5 and 1. For robot traffic across each Web server, we generate a plot of the R/S statistic on log-log scale and then estimate H as the slope of the linear regression curve fitting the data.

Periodogram

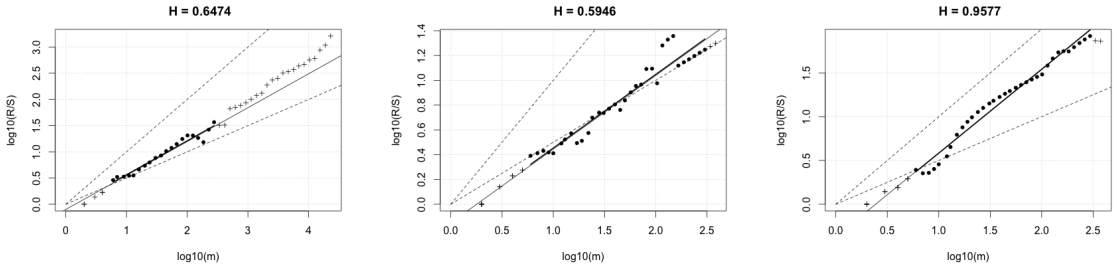
A periodogram is a plot of the spectral density of a time series against frequencies. The periodogram of a time series is defined by:

$$I(\lambda) = \frac{1}{2\pi N} \left| \sum_{j=1}^{N} X_j e^{ij\lambda} \right|^2$$

where λ is the frequency and $i = \sqrt{-1}$. As $\lambda \to 0$, on a log-log plot of the periodogram of an LRD time series will show a slinear trend with slope $\alpha - 1 = 1 - 2H$. Similar to the R/S plot, we fit a linear regression curve and estimate $\alpha - 1$ as the slope of the regression curve.

Wavelet Analysis

Wavelet analysis estimates the hurst parameter by taking a discrete wavelet transformation of the time series. For an LRD time series $X = \{X(t)|t \in \mathbb{R}\}$, this



(a) Coop **(b)** Roper Center **(c)** School of Engineering

Fig. 4.13: R/S plots of Web robot request arrivals

transform is given as:

$$D(j,k) = \int_{\mathbb{R}} X(t)\psi_{j,k}(t)dt$$

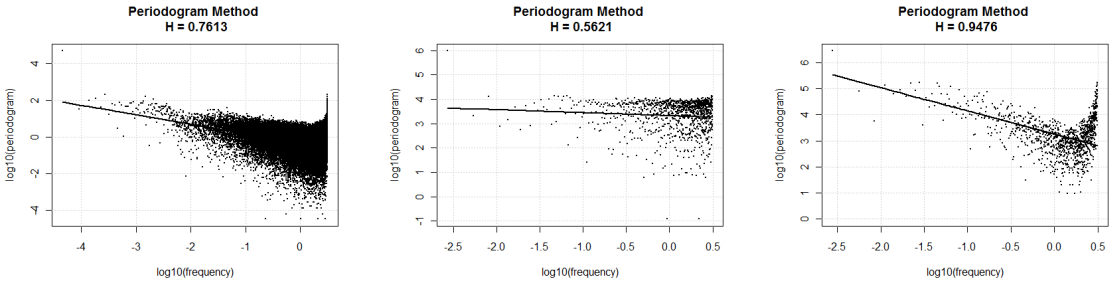
where $\psi_{j,k}(t) = 2^{-j/2}\psi(2^k t - k)$ is a child wavelet obtained by a shifting and dilation of the prototype wavelet function ψ. By keeping the child wavelet scaling parameter j fixed, we can rewrite $D(j,k)$ as $D(j,k) = 2^{j(H+1/2)}D(0,k)$. The expected value of the square of the log of this function is can be written as:

$$\mathbb{E}[\log D(j,k)^2] = j(2H+1) + \mathbb{E}[\log D(0,k)^2]$$

In other words, H can be estimated as the slope of the regression curve for $\mathbb{E}[\log D(j,k)^2]$ as the scale parameter j is varied.

Whittle estimator

The whittle estimator is a maximum likelihood estimate of the linear trend in the periodogram that also provides a confidence interval. This estimator requires a



(a) Coop **(b)** Roper Center **(c)** School of Engineering

Fig. 4.14: Periodogram of Web robot request arrivals

model of the underlying process to be specified a priori. This model can be either a Fractional Gaussian Noise (FGN) process with $1/2 < H < 1$ or a fractional ARIMA(p, d, q) (fARIMA) process with $0 < d < 1/2$. The main difference between the FGN and fARIMA models is that fARIMA assumes that short-range dependency also exists, but FGN does not. Thus, we used the Whittle estimator under the FGN model.

4.4.3 LRD analysis

To analyze whether or not robot traffic is LRD, we first plot the R/S statistic and periodogram of robot requests in Figures 4.13 and 4.14. The dotted lines in the R/S plots are boundaries that have a slope of 0.5 and 1, respectively. The plots are in agreement with their estimate of the hurst parameter on the SoE and Roper Center servers, and there is a small discrepency for the Coop server. In table 4.3, we additionally include the estimated hurst parameter values using the

wavelet and the Whittle estimator. For the Whittle estimator, a 95% confidence interval on the estimate is also included. The wavelet estimator for the Coop server more closely agrees with the periodogram estimate, and the value of the Whittle estimator and range of the 95% confidence intervals both agree with the estimates derived using the other methods. From this analysis, we find that Web robot traffic exhibits long range depdenence regardless of the domain of the server and the types of Web applications hosted.

Server	R/S	Periodogram	Wavelet	Whittle [95% CI]
Coop	0.65	0.76	0.74	0.73 [0.72,0.73]
Roper	0.59	0.56	0.51	0.53 [0.49,0.57]
SoE	0.96	0.95	0.89	0.99 [0.96,1.04]

Table 4.3: Summary of hurst parameter estimates

4.4.4 LRD generation

Another important question about the arrival process of robot requests is whether or not the long range dependence of Web robot traffic is generated by the same mechanisms as it is for all Web traffic. A well-accepted model that explains the generation of LRD traffic considers streams of Web traffic that alternate between ON and OFF periods, where the distribution of ON times is governed by the heavy-tailed response sizes with parameter α_{on} and OFF times are associated with the heavy-tailed inter-arrival times of requests with parameter α_{off} [27]. The aggregation of many such streams forms a self-similar fractional Gaussian

Server	α_{on}	α_{off}
Coop	3.88	3.68
Roper Center	2.25	3.33
SoE	2.28	3.69

Table 4.4: Heavy-tail parameters for ON/OFF periods

noise process with Hurst parameter $H = (3 - min(\alpha_{on}, \alpha_{off}))/2$. Since the distributions of ON times consistently show a heavier tail ($\alpha_{on} < \alpha_{off}$, generally, LRD in Web traffic is associated with the distribution of response sizes.

The distribution of response sizes for human traffic can be related to the heavy-tailed distribution of files hosted on a Web server. If we assume that all Web clients have a sufficiently large cache that is seldom emptied, the distribution of the sizes of all responses should approach the distribution of resources hosted on the server as the number of clients considered increases. Web robot traffic, however, is distinct because each robot may or may not take advantage of a client-side cache to store resources. Furthermore, robots may be unable to interact with Web applications the same way that humans do, preventing them from accessing the same type of information. Thus, it is likely that the LRD in robot traffic could instead be associated with the OFF time distribution (the inter-arrival times of requests) and not with the heavy-tailed distribution of the requested resource sizes.

To examine this further, Table 4.4 lists the values of α_{on} and α_{off} for robot traffic for each server. These estimates were obtained using Clauset et al. maxi-

mum likelihood method [23] across a range of values at which the heavy-tail can start, choosing α that best fits the empirical data according to the Kolmogorov-Smirnov goodness-of-fit statistic. Since $\alpha_{on} < \alpha_{off}$ for the Roper Center and SoE Web servers, LRD in robot traffic on these servers may be associated with the response size distribution. On the Coop server, however, we find that $\alpha_{off} < \alpha_{on}$. In other words, LRD in robot traffic over an e-commerce server may be generated by the distribution of inter- arrival times.

4.5 Related Research

Many studies have previously explored the characteristics of Web traffic using similar distributions for heavy- or power-tailed analysis. The Pareto distribution of response sizes was first observed by Crovella *et al.* [27] and has since been supported from probabilistic models by Mitzenmacher [88]. From these studies came results by Barford *et al.* and Arlitt *et al.* suggesting that the distribution of response sizes also exhibit power tails [13,7]. Campos *et al.* notes how even more precise fits can be obtained by using mixtures double Pareto-Lognormal distributions [55]. The interarrival times of human requests have been found by Crovella *et al.* to be power-tailed and characterized by a Weibull distribution [26]. These authors note that the distribution of interarrival times during ON periods is Weibull, while the OFF periods follow a Pareto distribution. When considering the interarrival times between all TCP packets across a link (with no ON/OFF

period differentiation), Downey suggests that the Pareto distribution fits well in the body but its extreme tail behavior does not follow power-tailed characteristics [39]. Previous work also suggests that the session request volume for all Web traffic is power-tailed. For example, Goseva-Popstojanova *et al.* and Menasce *et al.* have reported that the number of requests per session is well fitted by a Pareto distribution with $1.18 \leq \alpha \leq 3.94$ [47,87].

In the above studies, power-tailed behaviors were observed by considering a single test, and even by only a visual inspection of distribution fits on log-log scale. Our work however considered numerous tests together in order to more accurately evaluate whether or not a power-tailed trend exists. As suggested by our analysis of truncated data sets and fits that suggested the presence of a power tail when we could not conclude one exists, running multiple tests for power tail analysis is essential. Additionally, the above work considered did not consider Web robot traffic. Instead, robot traffic was filtered out in a preprocessing step, or was assumed to be sufficiently low in volume so as to not strong impact the work's findings. In this paper, we focused exclusively on Web robot traffic and discovered that many characteristics do not exhibit power-tailed behavior.

4.6 Chapter Summary

This chapter evaluated whether or not Web robot traffic exhibit power-tailed trends in many of their traffic characteristics and if the arrival process of their

requests to a Web server exhibits LRD. We used access logs collected from public Web servers that span three distinct domains and found that response sizes and session request volume does not exhibit power-tails, while interarrival and intersession times do. Our conclusions are based on a comprehensive analysis that used multiple tests to identify power-tailed behavior. Non-power-tailed response sizes and session request volumes may arise because robots may not maintain a client side cache, may schedule and send requests at a specific rate based on the tasks that they are programmed to do, target only the resources they are interested in during a site crawl, and follow widely varying navigational patterns. It is less surprising to find power-tailed interarrival and intersession times because there is a deep relationship between power-tailed arrival distributions and long-range dependence [37], which Web robot traffic exhibits as confirmed by our study.

Chapter 5

Predictive Caching for Web Robot Traffic

Web caches are the primary tool Web servers use to provide low response rates to client requests [122]. They also help reduce the number of bottlenecks on a network [2], and are instrumental for building scalable server clusters [93,100,18]. A Web cache is defined as a reserved amount of space reserved in a high level of a Web server's memory hierarchy available for the storage of resources. This high level memory store, which typically corresponds to a Web server's main memory or a specialized hardware device, is located closer to its central processing unit and thus offers faster read access to resources stored within them. Thus, should a client request a resource that is stored a Web cache, referred to as a *cache hit*, the server can service the request at a faster rate than if the resource was located outside of the cache (a *cache miss*). The performance of a cache is measured in terms of its hit ratio, which is defined as the proportion of requests that are made for resources stored in the cache [99]. High performance caches are desirable because cache misses increase the latency of a request and requires the server to perform costly I/O operations can gradually reduces the reliability of the server [84].

Ideally, a Web server would be able to store every resource that may be request by a client into its Web cache, however, hardware and cost barriers limit the amount of space in higher levels of a system's memory hierarchy. For example, pricing reports researched in 2014 finds the cost of main memory storage for a Web server to be approximately $9,000 per TB[1], versus just $51 per TB for hard disk storage[2]. Further constraints in the hardware architecture of a Web server limits the maximum amount of main memory to be supported, most of which must be reserved for running the Web server software and many of its supporting applications. Faced with the reality that one cannot simply build a cache that can store all resources on a Web server, numerous research studies have introduced methods designed to optimize the performance of restricted-size Web caches [1,2,10,13,65,122]. Broadly, these solutions are based on either new hardware and network architectures that improves the throughput of cache accesses and minimizes the cost of cache misses [102], or on software *polices*, which are algorithms that carefully decides what resources should be loaded, evicted, and admitted into a cache in a way that tries to maximize its hit ratio. Such polices may be defined by heuristic rules [64,71,65,19], or may incorporate predictive models that anticipate future requests given a past history [92,125,91].

Prevalent Web caching architectures and polices were developed without considering the possible impact that Web robot traffic, with its differentiated

[1] `http://www.jcmit.com/memoryprice.htm`

[2] `http://www.statisticbrain.com/average-cost-of-hard-drive-storage/`

functionality [33], access patterns [38], and traffic characteristics [32,35], may have on its performance. Previous studies have noted that a cache's hit ratio is reduced by Web robot traffic [4], supporting the notion that the intrinsic differences in their traffic make their requests incompatible with current Web caches. The effect of an incompatible cache may be very significant; previous studies find cache hit ratios to grow only logarithmically with cache size [17]. Thus, a reduction in a cache's hit ratio due to robot visits by even just a few percentage points reduces its performance to a cache with a higher hit ratio but whose size is many times smaller. To the best of our knowledge, a caching policy designed from the ground-up for servicing both robot and human traffic with high performance has not yet been proposed.

Given that robot traffic levels on the Internet have reached new heights and now account for over half of all HTTP requests submitted globally[3], caches that can support robot requests have never been more essential. This chapter proposes the design of such a cache, featuring a dual-caching architecture where requests submitted by robots or humans are serviced using separate caches. The caching policy for the robot cache uses an Elman neural network to predict the *type* of the next request following a sequence of previous request types. Based on this prediction, we load the Web robot cache with the most popular resources of the type predicted. We compare the predictive performance of an Elman neural

[3] `http://www.bbc.com/news/technology-25346235`

network against a number of other multinomial predictors and confirm that it offers strong predictive power. We evaluate our Web robot caching policy against a suite of baseline polices that encapsulates the traffic features that most polices used in practice are based on. Experimental results show that our robot caching policy outperforms the best baseline method by over 33% on a conservatively sized 40MB cache. Our policy's performance outstrips the baseline methods even further as the size of the Web cache increases.

This chapter is organized as follows. Section 5.1 describes the dual-caching architecture of our system. Section 5.2 explains our caching policy and how we predict resource types. Section 5.3 evaluates the performance of our caching algorithm against the baseline suite using streams of Web robot requests from the UConn SoE Web server. We summarize and conclude the chapter in Section 5.4.

5.1 Caching Architecture

The primary reason prevalent caching systems cannot service robot and human traffic with high performance may be because they route requests from both classes to a single, common cache. Robot and human traffic, however represent two very different profiles of traffic. Polices designed around the features of human traffic should thus not be expected to adequately service robot requests, and vice versa. Furthermore, there is a danger that a caching policy designed according to the characteristics of an unfiltered stream of requests that contains a (now realistic)

equal mix of robot and human traffic will fit itself to trends and behaviors of their interleaved request stream that does not reflect the patterns of either class. Such a cache would not perform well when servicing either robots or humans.

Because the traffic profiles of robots and humans are so different, it is intuitive that we should handle their requests *separately* by routing traffic from each class to a separate cache. This way, we can use a caching policy known to be compatible with human traffic [21,65] for one cache, and define a new policy that governs the behavior of a separate cache for robot requests. Figure 5.1 shows how, by using our method for offline and real-time detection [34] as described in Chapter 2, such a dual-cache architecture can be achieved. The offline detector will separate past requests made and recorded from the Web server's access log. The separated traffic can then be analyzed to learn the features of robot and human traffic separately. These features can then be applied to the development of the caching polices for the robot and human Web caches. The real-time detector will be responsible for routing a request to either the robot or human cache. In light of the many analyses that compare the relative merits of various caching polices over human requests [12,122,2], for the remainder of this study, we concentrate on the development and evaluation of a policy for the Web robot cache.

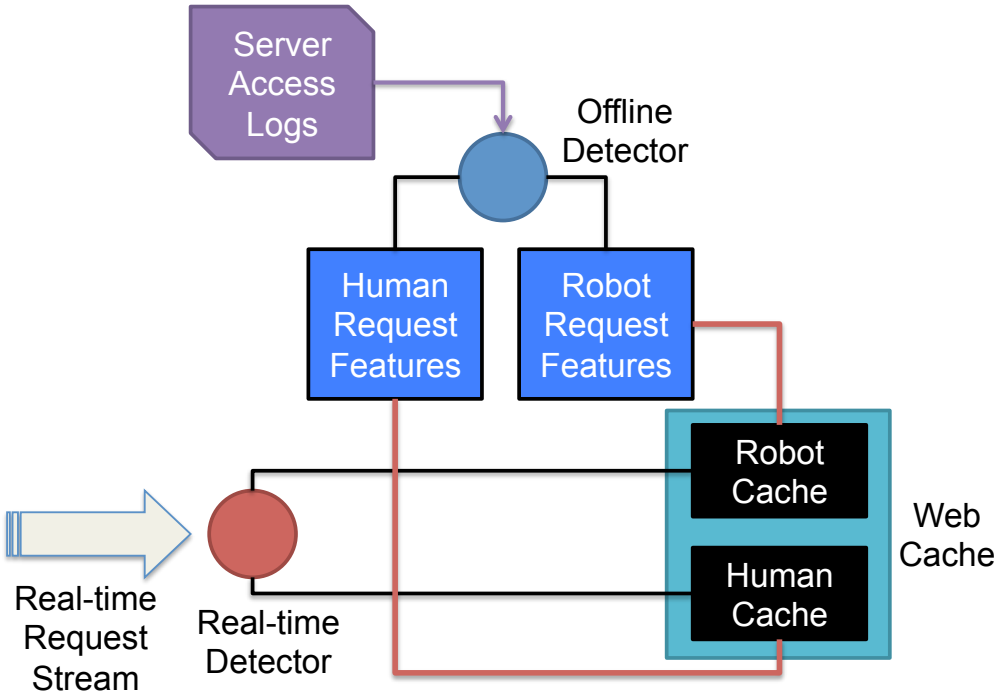


Fig. 5.1: Dual-cache architecture for robot and human requests

5.2 Caching Policy for Web Robots

Advanced caching polices exploit characteristics of Web traffic which are modulated by the way humans interact with a Web site. For example, the Web browser or device a human uses to submit requests will typically follow a request for an html file with a series of requests for the embedded resources on the page. This has led to the development of *link-distance* caching polices, which assume that the fewer links one resource is from the another recently requested, the more likely it is that it will be requested in the future and thus should be loaded into the Web cache [114]. Other polices assert that more recent accesses to a resource suggest

that it will be requested again in the future [64,89,71]. For Web servers that host Websites that most humans always enter through a home page, or those that host time-sensitive content such as news articles, status reports, or information about an event, such *least-recently-used* caching polices may perform well.

Web robots, however, are able to crawl a Web site in an unregulated fashion and request resources in any order, irrespective of the site's link structure. Furthermore, depending on the functionality of a robot, it may not be interested in harvesting time-sensitive information. Link-distance and least-recently-used caching polices are thus two examples of caching polices that depend on behaviors intrinsic to human traffic and thus is unlikely to perform well when faced with robot traffic. Thus, rather than adapting an existing policy tailored to human traffic, this section introduces a new caching policy that is based on the patterns of request types within sequences of Web robot requests. first give an overview of our proposed policy and then explain how our policy can learn and leverage the request type patterns of Web robot traffic.

5.2.1 Policy overview

Figure 5.2 describes our proposed policy for the Web robot cache. The key ingredients of the policy is: (i) predicting the *type* of the next request that will be made by a Web robot using a multinomial predictor; (ii) ordering resources of each type by how frequently they had been requested in the past; and (iii) always

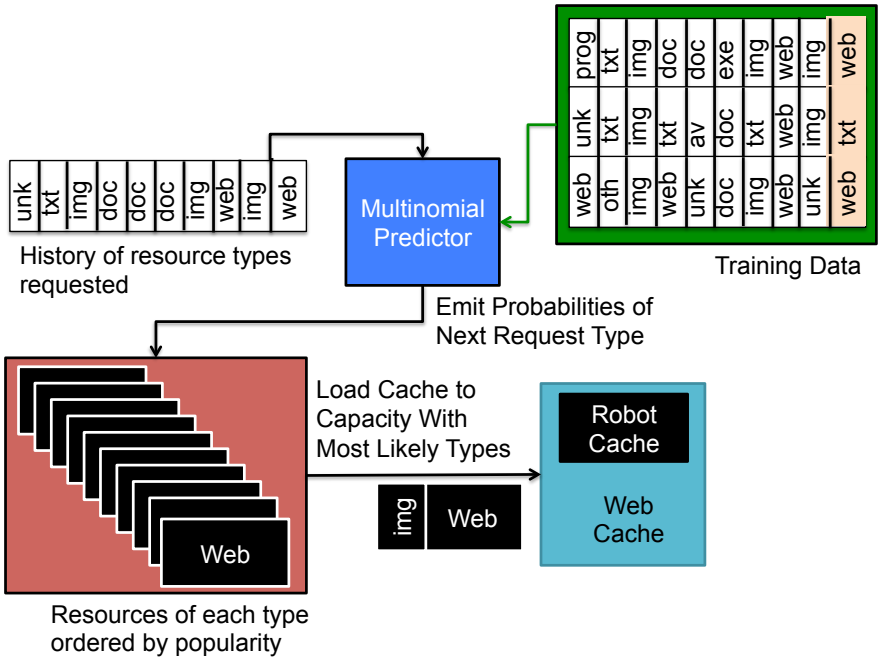


Fig. 5.2: Robot caching policy based on request type patterns

admitting the most popular resources across *all* resource types. We explain the rationale behind these three ingredients below.

Request type patterns

Our analysis of robot traffic in Sections 3.3 and 2.2.1 found that different types of robots have a penchant for requesting resources only of certain *types*. For example, Table 3.5 lists *Geniebot* as a robot that almost exclusively submits requests for resources of type *web* and *txt*, and *Asterias* as one that primarily makes *web* and *av* requests. Furthermore, these robot request types also exhibit a pattern within their ordering. For example, Table 2.3 shows how the *MJ12bot* only requests *web*, *txt*, and *noe* resources, and exhibits is a long string of requests for *web* following

a single request for *txt*. We may thus anticipate that, following a request for a *txt* or *web* resource, that the next request will be of type *web*. As another example, the table shows how when the robot *Yandex* is not requesting a *web* resource, it submits pairs of requests of the same type. Studying the types of resources requested and the patterns of these requests within sequences may thus help us anticipate the next type of resource that will be requested. We thus employ a multinomial predictor, trained over sequences of past robot request types from a Web server log file, that computes the probability that the next Web robot request will be of a certain type. We define these types as the same ones listed in Table 2.2, along with an *unk* type for requests made to an unknown type of resource.

Resource popularity

Numerous studies confirm that the frequency a resource is requested on a Web server follows a power-tailed distribution [50]. In other words, there exists a small subset of resources that are requested far more frequently than other resources on a Web server. This finding serves as the basis for many popularity-based caching polices [65,19,21] that exhibit high hit ratios. As illustrated in Figure 5.3, which plots the frequency all resources requested by Web robots between August 1st and September 17th 2009 from the UConn SoE Web server, we find that the frequency resources are requested by Web robots also exhibit a power-tail. Owing to the

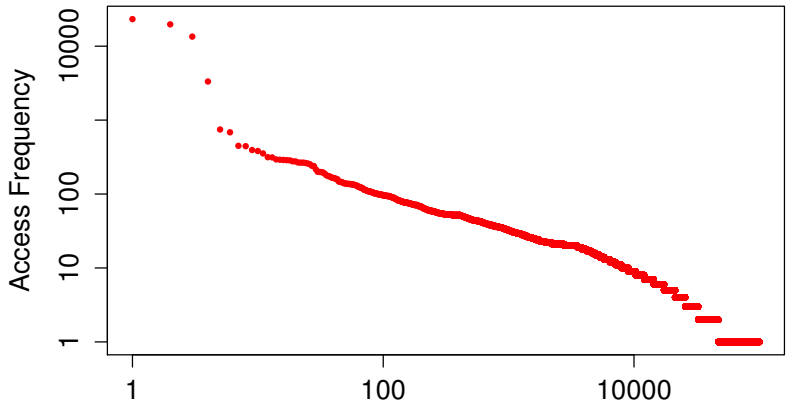


Fig. 5.3: Distribution of Web robot resource popularity

success of popularity-based caching in the literature and the power-tailed nature of resource popularity by Web robots, our policy loads into the Web robot cache the most frequently requested resources whose type is most likely to be requested next. If the total size of all resources of a given type is smaller than the size of the Web robot cache, we load all resources of the predicted type as well as the most popular resources of the next most likely resource type that will be requested.

Globally popular resources

Before loading the robot cache with the most popular resources of the type predicted, the policy first admits a small number of resources that are *globally* popular, irrespective of their type. This step lets the policy leverages the power-tailed property of resource popularity, where the most popular resources are requested

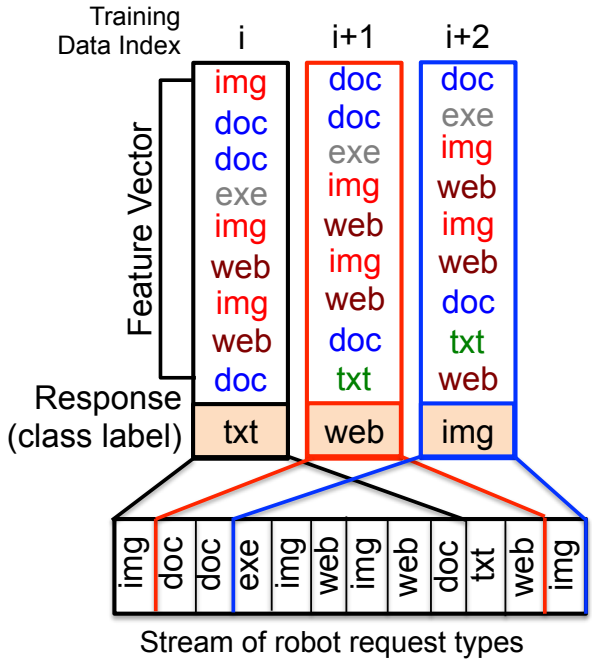


Fig. 5.4: Feature vectors extracted from a stream of robot requests

a disproportionately higher number of times. For example, Figure 5.3 illustrates how the three most popular resources are requested an order of magnitude more frequently than the fourth most popular resource. Furthermore, the ten most popular resources are collectively requested 12.9% of the time. By reserving a small portion of the robot cache just for these hyper-popular resources, our policy thus guarantees cache hits for a substantial proportion of requests.

5.2.2 Learning request type patterns

For the purpose of building a training data set for the multinomial predictor, we define a request type sequence as an ordered sequence of the types of n consecutive

requests $(x_1, ...x_n)$ made during a Web robot session. A record of the training set $\mathbf{r}_i = (v_i, l_i)$ is given by the feature vector $v_i = (x_1, ...x_{n-1})$ and class label $l_i = x_n$. Thus, the multinomial predictor will identify patterns within the first $n-1$ requests and associate them with the type of the n^{th} request during training. The extraction of request type sequences from Web robot sessions is illustrated further in Figure 5.4. In the figure, sequences of size $n = 10$ are extracted from a stream of robot requests. From a 12 requests in the stream, 3 training records can be constructed. The feature vector of the first record is composed of the first nine and its class label is given by the tenth request; the feature vector of the second record has the second through tenth request and its class label is given by the eleventh request; and the third record has a feature vector given by the third through eleventh request and its class label is the type of the twelfth request. Once trained, the predictor will maintain a a history of the types of the last $n-1$ requests submitted by a Web robot to the server and, based on this history, generate the probability distribution for the next type of request that will be made. Next, we discuss the selection and training of our multinomial predictor.

Prediction model

A number of machine learning models for multinomial prediction, such logistic regression ensembles [49], multinomial naive bayesian networks [72], support vector machines [111], random forests [29], or feed-forward neural networks [58], may be

capable of predicting request types.

As mentioned in our discussion from Section 5.2.1, we recall that both *features* of a request sequence as well as the *order* of types both carry important information that may be useful for predicting the type of the subsequent request. For example, consider the two request sequences $(web, web, img, img, web)$ and $(web, img, web, img, web)$. On the one hand, both sequences exhibit similar features, namely, that they are composed of 3 *web* and 2 *img* requests. These features may imply that the next request will be either for a *web* or *img* resource. But by incorporating order within the sequences as well, we may predict that the type of the next request will be *web* in the first pattern, and *img* in the second.

The multinomial predictor for our problem should thus be capable of finding associations between not only the content, but also between the order of content and the class labels of feature vectors. A particularly suitable model then is an *Elman Neural Network* (ENN) [42]. An ENN, as seen in the top of Figure 5.5, is a feed-forward neural network with a single hidden layer (blue). Each hidden unit corresponds to a logistic function whose parameters are fitted so that the function can identify the presence of an implicit pattern or characteristic in the input data. For example, the training phase may fit the parameters of a hidden unit layer to 'activate', that is, return a value close to 1, when a request sequence contains only *web* or *img* requests. Another, separate hidden unit may activate if *unk* requests are dominant in the sequence.

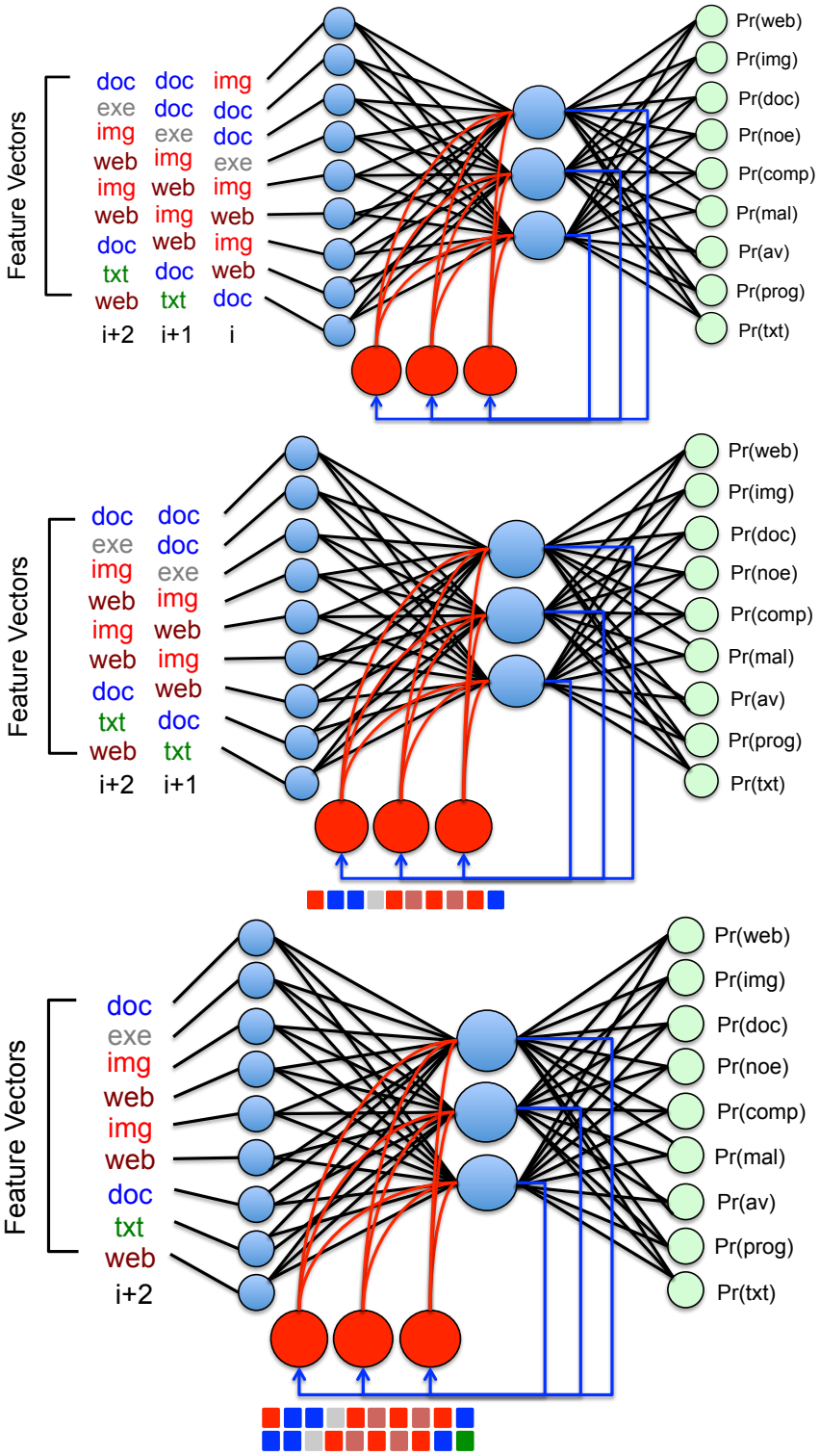


Fig. 5.5: Elman Neural Network training over request type sequences

Importantly, an ENN is augmented with an additional *context* layer (the red units in Figure 5.5) that, during the training phase, stores the output of hidden layer nodes that are activated during the training process. These values are fed back into the same hidden layer node during training of the subsequent request. This recurrence effectively tunes a hidden unit's parameters to not only the features of a training record, but also to the ordering of requests in previous feature vectors. To conceptualize this, consider the diagram of an ENN with three hidden states in Figure 5.5. During training on record i, the hidden units use i's feature vector to emit a value based on their present parameters, which are routed to the output units as well as to a context unit. The hidden unit parameters are then adjusted according to the error between the networks prediction and the actual label of feature vector i. When the network begins its learning process for training sample $i+1$ in the center diagram, the hidden nodes now emit a value based on the feature vector v_{i+1} of training record $\mathbf{r}_{i+1}$ as well as its previous activation value from training sample i. Since this previous activation value represents a hidden unit's computation from when an *img* type in the first position of feature vector v_i, a *doc* in the second position of v_i, and so forth, the hidden node's new value will consider the state of previous request sequences as well as the feature vector of $\mathbf{r}_{i+1}$. Similarly, when the ENN begins training over record $\mathbf{r}_{i+2}$ in the bottom diagram of Figure 5.5, the context layer incorporates the state of the hidden units from when the previous two training vectors were analyzed.

5.2.3 Model evaluation

We evaluated how well an ENN can predict the type of a robot request using a sample of all Web robot requests to the UConn SoE Web server between August 1st and September 17th 2009. Our offline detection algorithm was used to extract a total of 487,972 robot requests during this time period. We ordered these requests by the time they were submitted, and grouped them into robot sessions with a 30 minute timeout. From each session, we extracted all requests sequences of length $k = 11$. Our choice of k was based on our robot detection study, which found that sequences $k \geq 10$ request types was sufficiently long to identify if it was generated by a robot or a human [34]. After filtering all robot sessions of length less than 11, we collected 362,390 sequences of robot type requests.

ENN network size

Importantly, the number of hidden units that should be used in an ENN must be specified *a priori*. This hyper-parameter needs to be carefully selected; large numbers of hidden units lets the model make predictions using more features of request sequences, but require a huge amount of computational power to train [124] and increases the risk of overfitting the training data. To find the best number of hidden units for our model, we took sequences of requests made during the first four and a half weeks (approximately 60% of our sample) and trained a variety of ENNs sporting between 2 and 40 hidden units. Figure 5.6 reports the mean

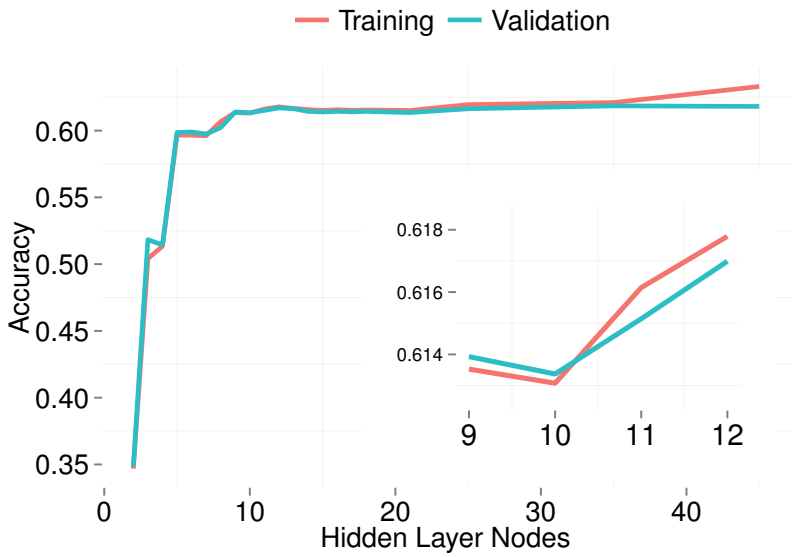


Fig. 5.6: Mean training and validation error vs. ENN hidden layer size

training and validation of these ENNs derived using 10-fold cross-validation. The model's accuracy over both training and validation data does not improve appreciably beyond $n = 10$ hidden layer units. Furthermore, for $n > 10$, the ENN shows symptoms of overfitting the training data as its accuracy over the training folds rise above the validation fold. Thus, we find an ENN with 10 hidden units to be suitable for our prediction task.

Prediction evaluation

We used the ENN model trained from the first four and a half weeks to predict the type of the request following the request type sequences in the remaining three weeks (approximately 40% of the sample). We compared its accuracy against

two baseline models using the same training data, namely multinomial logistic regression (MLR) and a first-order discrete time Markov chain (DTMC). The MLR model consists of an ensemble of binary logistic regression classifiers, each of which is trained to emit the probability that the class label of a feature vector is of a specific resource type. Thus, the number of classifiers in the model is equal to the number of possible classes (resource types). For prediction, MLR runs all classifiers with the same data, identifies the one that returns the highest probability, and returns the request type it corresponds to. The DTMC model computes $p_{i,j}$, defined as the proportion of time a type j request follows a type i request, for all resource types i and j in the training data. It subsequently predicts that the next request will be of type $\hat{j}(i)$, where i is the type of the last request made, as $\hat{j}(i) = \arg\max_j p_{i,j}$. We selected these two baselines because they each correspond to one of the two important qualities of the sequences that we based our choice of an ENN on: MLR predicts according to only the *features* of sequences, while the DTMC just considers the *order* of request types. We can thus justify the use of an ENN, which combines both of these features in a single model, if its accuracy is stronger than these baselines.

Table 5.1 lists the accuracy of these three predictors. The left column gives the absolute accuracy, while the Gain-RG column lists the percent improvement of the method compared making a random guess of the next request type, which we assume to be equal to $1/m$ where $m = 11$ is the number resource type classes.

Model	Accuracy	Gain-RG	Gain-MLR	Gain-DTMC
Random Guess	**0.091**	-	-	-
MLR	**0.338**	70.41%	-	-
DTMC	**0.392**	74.49%	16.0%	-
ENN	**0.647**	84.54%	47.8%	39.4%

Table 5.1: Accuracy of different request type predictors

Gain-MLR, Gain-DTMC, and Gain-ENN lists the percent improvement in using a method compared to MLR, DTMC, and ENN, respectively. The Gain-RG of both MLR and DTMC is over 70%, suggesting that both the features of a request sequence as well as sequence order are similarly important for predicting request types. However, request order may be a better tell of the type of the next request compared to a sequence's features, since the DMTC model is 16% more accurate than MLR. ENN combines sequence features and order into a single model to achieve an even higher Gain-RG of 84.54% and sports a classification accuracy 0.647. This combining gives the ENN much stronger performance compared to the MLR (47.8% more accurate) and DTMC (39.4% more accurate) models.

5.3 Caching Policy Evaluation

In this section, we evaluate the performance of our ENN caching policy for Web robot requests. We use the same stream of robot requests used to evaluate the performance of the ENN predictor to simulate the operation of a cache with sizes that varies between 1MB and 40MB. We selected a number of other caching polices

to compare the hit ratio of the ENN predictor against. These polices are each rooted in identifying the a basic property of Web traffic that the majority of the most popular polices used in practice are based on [12,2]. The other polices are:

- **Least-Recently-Used (LRU)**: This policy always keeps the resources most recently requested in the cache. If a request arrives for a resource not in the cache, the policy evicts the resource least recently used and replaces it with the resource requested.
- **Log-size**: This policy first loads the cache with the most recently used resources. On a cache miss, the policy always chooses to evict the resource with the largest size. Ties are broken by comparing the floor of the log of the resource sizes. If the resource requested on a cache miss is larger than any resource in the cache, it does not get admitted.
- **Popularity**: This policy loads the cache with the most frequently requested resources. Following every request, the policy checks if a resource has become more popular than the least popular cached resource. If so, that resource is replaced.
- **Hyper-G** [1]: Hyper-G is a hybrid caching policy that considers both the time since last access and popularity of a resource. On a cache miss, the policy evicts the resource that has been requested the fewest number of times in the cache. LRU is used to decide between resources that have been requested the same number of times.

Model	1MB	2MB	3MB	4MB	5MB	8MB	12MB	20MB	40MB
Lg-size	0.055	0.056	0.057	0.057	0.057	0.058	0.058	0.059	0.059
LRU	0.111	0.126	0.136	0.141	0.145	0.153	0.159	0.165	0.175
Hyp-G	0.174	0.178	0.172	0.180	0.176	0.188	0.189	0.212	0.236
Pop	**0.192**	**0.204**	0.206	0.205	0.205	0.205	0.223	0.224	0.282
ENN	0.185	0.199	**0.212**	**0.220**	**0.228**	0.**258**	**0.284**	**0.335**	**0.425**
% Gain	-3.4%	-2.5%	3.78%	6.82%	10.1%	20.5%	21.5%	33.1%	33.6%

Table 5.2: Hit ratio of ENN-based cache policy compared to baselines

Table 5.2 list the hit ratio of each policy under caches of different sizes. Except for very small cache sizes (2MB or less), the ENN based caching policy outperforms all of the baseline methods, with an improvement in hit ratio that grows from just 0.005 for a 3MB cache up over 0.143 using a 40MB cache. Recalling that hit ratios only grow logarithmically with cache size, our policy for caching robot requests thus offers the same performance as using an exponentially larger cache paired with the best baseline method. The table also lists the percent improvement in hit ratio for our ENN policy against the highest performing policy among the others, showing how the larger the size of the cache available for robot requests, the greater the improvement offered by our ENN policy.

We verified that the ENN policy consistently outperforms the others over different streams of robot requests by comparing policy hit ratios across 100 non-overlapping partitions of the test set. These partitions were created by dividing the test data into equally sized groups of contiguous requests. Figure 5.7 plots the trend of the hit ratio across these polices under different robot cache sizes. When the cache is 2MB or smaller in size, Hyper-G occasionally outperforms the

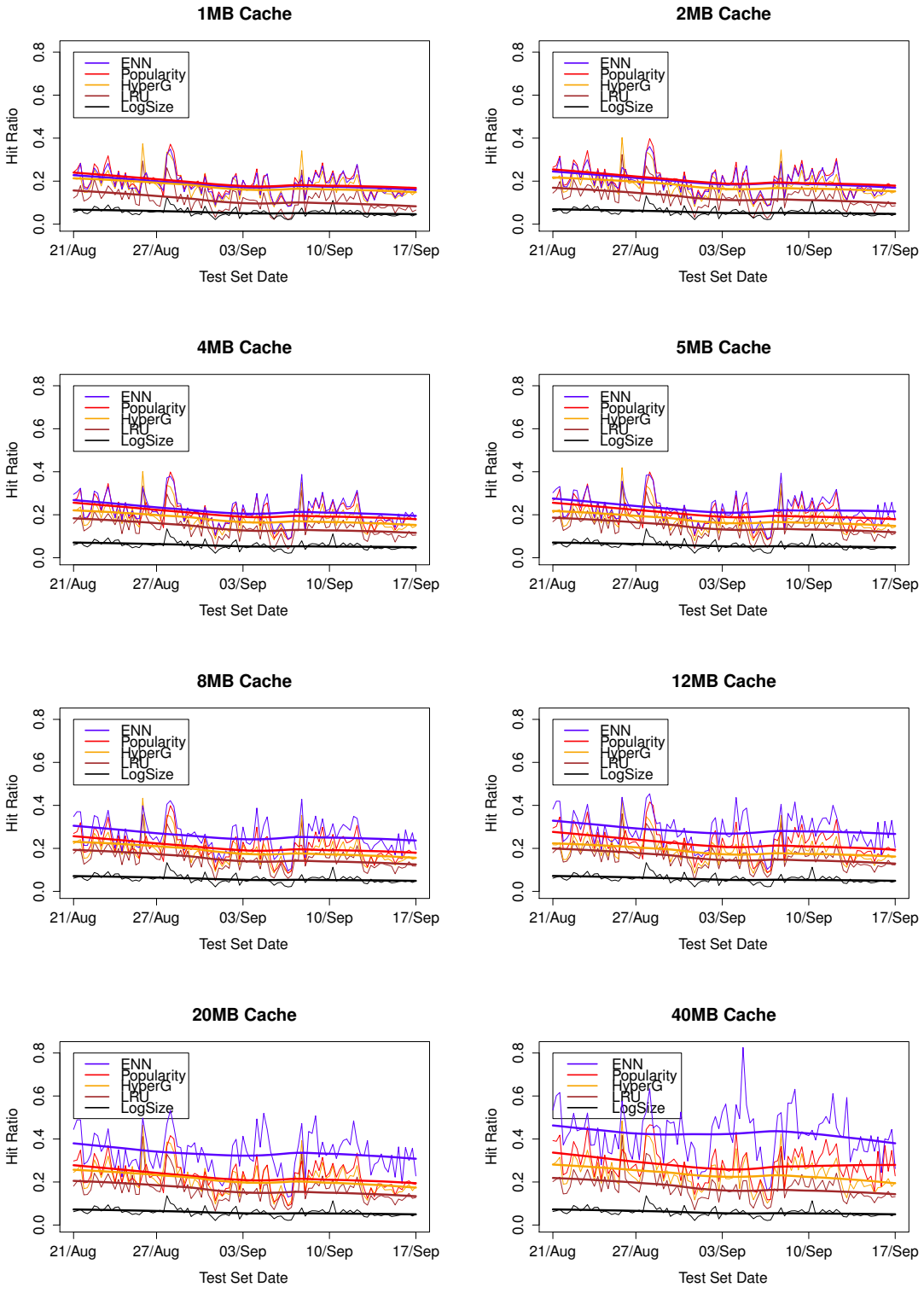


Fig. 5.7: Cache hit ratios over different sized caches

Popularity and ENN polices. Once we set the cache size to 4MB, however, the hit ratio of ENN begins differentiate itself as being higher for both Popularity and Hyper-G across almost every test set. This difference in hit ratio continues to grow higher as we increase the size of the Web cache.

5.4 Chapter Summary

This chapter presented the design of a novel dual-caching architecture and Web robot caching policy that can service requests with high performance. The key components of our architecture include the offline and real-time Web robot detectors described in Chapter 2 and an Elman neural network that can predict the next type of resource that will be requested given the history of previous request types. We experimentally verified that the ENN offers more predictive power than other multinomial predictors by considering both the features of and order of requests within a sequence. We compared the hit ratios attained by our ENN-based robot caching policy against a number of typically used polices. Except for very small cache sizes, our ENN-based cache gives the highest hit ratio. ENN yields progressively better percent gains over the best other policy (Popularity) as the size of the robot cache increases.

Chapter 6

Concluding Remarks and Future Directions

This dissertation presented a comprehensive survey of Web robot detection techniques, presented state-of-the-art methods for identifying their traffic on the Internet, comprehensively examined robot along multiple dimensions, identified essential differences in the statistical patterns of robot and human traffic, and introduced a novel caching architecture and policy that lets a Web server service robot requests at a much higher level of performance compared to a number of common caching techniques. The specific contributions of this dissertation are:

- **Survey and classification of robot detection methods**: Our detection survey dichotomized robot detection algorithms according to their fundamental detection philosophy. We used this classification to critically analyze and compare the existing work.
- **State-of-the-art offline and real-time robot detection**: Our detection approach addresses the key limitations revealed through our analysis by rooting analytical models by an intrinsic difference between robot and

human behavior. Experimental results verified that our approach can be used both offline and in real-time, with very high accuracy and reliability.

- **Multi-dimensional classification of Web robots**: A novel framework for classifying Web robots along multiple dimensions was introduced. In the framework, we divide robot visitors to a Web server by their functionality, workload characteristics, and preferred types of resources. These orthogonal dimensions allowed us to aggregate robot requests in ways that revealed new information about their behaviors.
- **Workload analysis of Web robots**: This work throughly analyzed the statistical features of robot requests across a number of Web servers in different domains. Through our analysis, we found that impotant traffic features, namely response sizes and session request volumes, are not power-tailed. However, the arrival process of robot requests do exhibit long-range dependence, similar to the arrival process of human requests.
- **New caching systems for improving Web server performance**: The design of a novel dual-caching architecture for servicing Web robot and human traffic with high performance was introduced. The caching system, which is built off of the lessons learned in the resource-type classification of Web robots and relies critically on the offline and real-time detection algorithms presented, uses an Elman neural network to predict resource request types.

Taken together, these contributes enable advanced studies of Web robot traffic (via offline detection), improves a Web servers ability to react to robot visits (via real-time detection), advances our understanding about the types of robots that crawl the Web and how they statistically compare to human traffic, and presents a method for improving Web server performance faced with ever-increasing levels of robot traffic.

further advances in our understanding of robot traffic and preparing Web servers to handle their traffic will be done along a number of future research directions. Specifically, future work will closely study the performance of our offline and real-time detector on Web servers across a variety of ways, and will develop plug-ins for common Web server platforms (IIS, Apache) so that real-time detection can be utilized in practice. Towards better understanding the statistical nature of robot traffic on the Web, deeper explorations of the intuitive arguments presented about why some aspects of robot traffic are not-power tailed will be done. Similar, more comprehensive investigations behind why long-range dependence in Web robot arrival processes depend on inter-arrival times across e-commerce servers, but not in an academic server. Finally, different of feature engineering appraoches that can improve the predictive power of our Elman neural network for robot caching will be investigated. This investigation may include the effect of training on longer request sequences and the addition of features such as resource sizes, time between requests, and the http response codes to also learn

over. Future work will also verify that our caching model performs well using traces of robot data from Web servers that span a number of different domains.

Bibliography

[1] M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox, and S. Williams. Removal policies in network caches for world-wide web documents. In *Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '96, pages 293–305, New York, NY, USA, 1996. ACM.

[2] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, 1999.

[3] L. V. Ahn, M. Blum, and J. Langford. Captcha: Using hard ai problems for security. In *In Proceedings of Eurocrypt*, pages 294–311. Springer-Verlag, 2003.

[4] V. Almeida, D. A. Menascé, R. Riedi, F. P. Ribeiro, R. Fonseca, and W. Meira, Jr. Analyzing Web robots and their impact on caching. In *Proc. Sixth Workshop on Web Caching and Content Distribution*, pages 299–310, 2001.

[5] S. Anbukodi and K. Manickam. Reducing web crawler overhead using mobile crawler. In *Emerging Trends in Electrical and Computer Technology (ICETECT), 2011 International Conference on*, pages 926 –932, march 2011.

[6] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical report, Hewlett-Packard, 1999.

[7] M. F. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. *ACM SIGMETRICS Performance Evaluation Review*, pages 126–137, 1996.

[8] M. F. Arlitt and C. L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, 1997.

[9] AWStats. Free log file analyzer for advanced statistics (GNU GPL), 2011. `http://awstats.sourceforge.net/`.

[10] H. Bahn, K. Koh, S. L. Min, and S. H. Noh. Efficient replacement of nonuniform objects in web caches. *Computer*, 35:65–73, June 2002.

[11] P. Bak. *How Nature Works: The Science of Self-Organized Criticality*. Springer-Verlag Telos, 1st edition, April 1999.

[12] A. Balamash and M. Krunz. An overview of web caching replacement algorithms. *Communications Surveys & Tutorials, IEEE*, 6(2):44–56, 2004.

[13] P. Barford, A. Bestavros, A. Bradley, and M. Crovella. Changes in web client access patterns - characteristics and caching implications. *World Wide Web*, 2:15–28, 1999.

[14] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Measurement and Modeling of Computer Systems*, pages 151–160, 1998.

[15] C. Bomhardt, W. Gaul, and L. Schmidt-Thieme. Web robot detection - preprocessing web logfiles for robot detection. *New Developments in Classification and Data Analysis*, pages 113–124, 2005.

[16] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu. The socialbot network: when bots socialize for fame and money. In *Proc. of 27th Annual Computer Security Applications Conference*, pages 93–102, 2011.

[17] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Usenix symposium on internet technologies and systems*, pages 193–206, 1997.

[18] V. Cardellini, M. Colajanni, and S. Y. Philip. Dynamic load balancing on web-server systems. *IEEE Internet computing*, 3(3):28–39, 1999.

[19] X. Chen and X. Zhang. A popularity-based prediction model for web prefetching. *Computer*, 36(3):63–70, 2003.

[20] S. Cheng and L. Peng. Confidence intervals for the Tail Index. *Bernoulli*, 7(5):751–760, 2001.

[21] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack. WAVE: Popularity-based and collaborative in-network caching for content-oriented networks. In *IEEE Conference on Computer Communications Workshops*, pages 316–321, 2012.

[22] H.-K. Choi and J. O. Limb. A behavioral model of web traffic. In *Seventh International Conference on Network Protocols*, pages 327–334, 1999.

[23] A. Clauset, C. R. Shalizi, and M. Newman. Power-Law Distributions in Empirical Data. Technical report, arXiv:0706.1062v2 [physics.data-an], 2009.

[24] R. Clegg. A practical guide to measuring the Hurst parameter. Technical Report CSTR-916, University of Newcastle, 2006.

[25] F. Clementi, T. Di Matteo, and M. Gallegati. The power-law tail exponent of income distributions. *Physica A: Statistical and Theoretical Physics*, 370:49–53, 2006.

[26] M. Crovella. Performance characteristics of the world wide web. *Performance Evaluation: Origins and Directions*, pages 219–232, 2000.

[27] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[28] Y. Deng, X. Meng, and J. Zhou. Self-similarity: Behind workload reshaping and prediction. *Future Generation Computer Systems*, 28(2):350–357, 2011.

[29] R. Díaz-Uriarte and S. A. De Andres. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):3, 2006.

[30] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou. An investigation of Web crawler behavior: characterization and metrics. *Computer Communications*, 28(8):880–897, 2005.

[31] D. Doran and S. Gokhale. Discovering New Trends in Web Robot Traffic Through Functional Classification. In *Proc. IEEE International Symposium on Network Computing and Applications*, pages 275–278, Cambridge, MA, 2008.

[32] D. Doran and S. Gokhale. Classifying Web robots by K-means clustering. In *Proc. of Intl. Conference on Software Engineering and Knowledge Engineering*, pages 97–102, Boston, MA, July 2009.

[33] D. Doran and S. Gokhale. A classification framework for web robots. *Journal of American Society of Information Science and Technology*, 63:2549–2554, 2012.

[34] D. Doran and S. Gokhale. Detecting Web Robots Using Resource Request Patterns. In *Proc. of Intl. Conference on Machine Learning and Applications*, pages 7–12, 2012.

[35] D. Doran and S. S. Gokhale. Searching For Heavy-Tails in Web Robot Traffic. In *Proc. of 7th IEEE Intl. Conference on Quantitative Evaluation of Systems*, pages 282–291, 2010.

[36] D. Doran and S. S. Gokhale. Web robot detection techniques: Overview and limitations. *Data Mining and Knowledge Discovery*, 22(1):183–210, 2011.

[37] D. Doran and S. S. Gokhale. Long Range Dependence in the Arrival Process of Web Robots. In *Proc. of Intl. Conference on Intelligent Network and Computing*, 2012.

[38] D. Doran, K. Morillo, and S. Gokhale. A Comparison of Web Robot and Human Requests. In *Proc. of ACM/IEEE Conference on Advances in Social Network Analysis and Mining*, pages 1374–1380, 2013.

[39] A. B. Downey. The structural cause of file size distributions. *ACM SIGMETRICS Performance Evaluation Review*, 29(1):328–329, 2001.

[40] A. B. Downey. Lognormal and pareto distributions in the internet. *Computer Communications*, 28(7):790–801, 2005.

[41] O. Duskin and D. G. Feitelson. Distinguishing humans from robots in web search logs: preliminary results using query rates and intervals. In *Proc. of 2009 workshop on Web Search Click Data*, pages 15–19, Barcelona, Spain, 2009. ACM.

[42] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[43] A. P. Gasch and M. B. Eisen. Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. *Genome Biol*, 3(11), October 2002.

[44] N. Geens, J. Juysmans, and J. Vanthienen. Evaluation of web robot discovery techniques: A benchmarking study. *Lecture Notes in Computer Science*, 4065/2006:121–130, 2006.

[45] C. L. Giles, Y. Sun, and I. G. Councill. Measuring The Web Crawler Ethics. In *Proc. of 19th Intl. World Wide Web Conference*, pages 1101–1102, 2010.

[46] K. Goseva-Popstojanova, F. Li, and A. Sangle. A contribution towards solving the web workload puzzle. In *Proc. of Intl. Conference on Dependable Systems and Networks (DSN 06)*, pages 505–516, Washington, DC, USA, 2006. IEEE Computer Society.

[47] K. Goseva-Popstojanova, A. D. Singh, S. Mazimadar, and F. Li. Empirical characterization of session-based workload and reliability for web servers. *Empirical Software Engineering*, 11:71–117, 2006.

[48] R. Gossweilier, M. Kamvar, and S. Baluja. What's up captcha?: a captcha based on image orientation. In *Proc. of 18th Intl. Conference on World wide web*, pages 841–850, 2009.

[49] M. Green, J. Björk, J. Forberg, U. Ekelund, L. Edenbrandt, and M. Ohlsson. Comparison between neural networks and multiple logistic regression to predict acute coronary syndrome in the emergency room. *Artificial intelligence in medicine*, 38(3):305–318, 2006.

[50] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang. Does internet media traffic really follow zipf-like distribution? In *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 359–360, New York, NY, USA, 2007. ACM.

[51] W. Guo, S. Ju, and Y. Gu. Web robot detection techniques based on statistics of their requested URL resources. In *Proc. of the 9th Intl. Conference on Computer Supported Cooperative Work in Design*, pages 302–306, 2005.

[52] H. Gupta, A. Mahanti, and V. Ribeiro. Revisiting Coexistence of Poissonity and Self-Similarity in Internet Traffic. In *Proc. of 17th IEEE/ACM Intl. Symposium on Modelling, Analysis, and Simulation of Computer Systems*, pages 1–10, 2009.

[53] X. He, H. Zha, C. H. Ding, and H. D. Simon. Web document clustering using hyperlink structures. *Computational Statistics & Data Analysis*, 41(1):19–45, 2002.

[54] F. Hernández-Campos, K. Jeffay, and F. D. Smith. Tracking the evolution of web traffic: 1995-2003. In *Proc of 11th IEEE Intl. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 03)*, pages 16–25, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[55] F. Hernández-Campos, J. S. Marron, F. D. Smith, and G. Samorodnitsky. Variable heavy tailed durations in internet traffic, part I: Understanding heavy tails. In *Proc. of 10th IEEE Intl. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 02)*, page 43, Los Alamitos, CA, USA, 2002.

[56] E. Hernandez-Orallo and J. Vila-Carbó. Web server performance analysis using histogram workload models. *Computer Networks*, 53(15):2727–2739, 2009.

[57] H. Hochheiser and B. Shneiderman. Using interactive visualizations of WWW log data to characterize access patterns and inform site design. *Journal of American Society of Information Science and Technology*, 52(4):331–343, 2001.

[58] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[59] D. Horowitz and S. D. Kamvar. The anatomy of a large-scale social search engine. In *Proc. of 19th Intl. World Wide Web Conference*, pages 431–440, 2010.

[60] P. Huntington, D. Nicholas, and H. R. Jamali. Web robot detection in the scholarly information environment. *Journal of Information Science*, 34(5):726–741, 2008.

[61] A. K. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 2:85–100, 1999.

[62] B. J. Jansen, A. Spink, and T. Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Inf. Process. Manage.*, 36(2):207–227, 2000.

[63] G. Jawaheer and P. Kostkova. Web Crawlers on a Health Related Portal: Detection, Characterisation, and Implications. In *Proc. of Developments in E-systems Engineering*, pages 24–29, 2011.

[64] P. R. Jelenković and A. Radovanović. Least-recently-used caching with dependent requests. *Theoretical computer science*, 326(1):293–327, 2004.

[65] S. Jin and A. Bestavros. Popularity-aware greedy dual-size web proxy caching algorithms. In *Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, ICDCS '00, pages 254–, Washington, DC, USA, 2000. IEEE Computer Society.

[66] P. Jourlin, R. Deveaud, E. Sanjuan-Ibekwe, J.-M. Francony, and F. Papa. Design, implementation and experiment of a YeSQL Web Crawler. In *Proc. of ACM SIGIR Workshop on Open Source Information Retrieval*, pages 56–59, 2012.

[67] T. Kabe and M. Miyazaki. Determining WWW user agents from server access log. In *Proc. Seventh Int'l Conference on Parallel and Distributed Systems*, pages 173–178, 2000.

[68] S. D. Kamvar and J. Harris. We feel fine and searching the emotional web. In *Proceedings of the fourth ACM Intl. conference on Web search and data mining*, WSDM '11, pages 117–126, New York, NY, USA, 2011. ACM.

[69] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: surviving organized ddos attacks that mimic flash crowds. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 287–300, Berkeley, CA, USA, 2005. USENIX Association.

[70] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 1990.

[71] R. E. A. Khayari, R. Sadre, and B. R. Haverkort. A class-based least-recently used caching algorithm for www proxies. In *Proceedings of the 2nd Polish-German Teletraffic Symposium*, pages 295–306, 2002.

[72] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes. Multinomial naive bayes for text categorization revisited. In *Advances in Artificial Intelligence*, pages 488–499. Springer, 2005.

[73] K. A. Kluever and R. Zanibbi. Video captchas: Usability vs. security. In *Proc. of IEEE Western New York Image Processing Workshop*, 2008.

[74] M. Koster. Guidelines for robot writers, 1993. `http://www.robotstxt.org/wc/guidelines.html`.

[75] M. Koster. A standard for robot exclusion, 1994. `http://www.robotstxt.org/orig.html`.

[76] R. Levering and M. Cutler. The portrait of a common html web page. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, pages 198–204, New York, NY, USA, 2006. ACM.

[77] F. Li, K. Goseva-Popstojanova, and A. Ross. Discovering web workload characteristics through cluster analysis. In *Proc. IEEE International Symposium on Network Computing and Applications*, pages 61–68, 2007.

[78] H. W. Lilliefors. On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318):399–402, 1967.

[79] X. Lin, L. Quan, and H. Wu. An automatic scheme to categorize user sessions in modern http traffic. In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM 08)*, pages 1–6, New Orleans, LO, November 2008.

[80] L. Lipsky. *Queueing Theory: A Linear Algebraic Approach*. Springer-Verlag, 2nd edition, 2009.

[81] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh. Online response time optimization of apache web server. In *International Workshop on Quality of Service*, pages 461–478. Springer, 2003.

[82] W.-Z. Lu and S.-Z. Yu. Web robot detection based on hidden Markov model. In *Proc. Intl. Conference on Communications, Circuits and Systems*, pages 1806–1810, 2006.

[83] B. A. Mah. An empirical model of http network traffic. In *Proc. of IEEE Intl. Conference on Computer Communications*, page 592, 1997.

[84] R. Matias et al. An experimental study on software aging and rejuvenation in web servers. In *International Computer Software and Applications Conference*, volume 1, pages 189–196, 2006.

[85] F. McCown and M. L. Nelson. Evaluation of crawling policies for a web-repository crawler. In *Proc. of Intl. Conference on Hypertext and hypermedia*, pages 157–168, New York, NY, USA, 2006. ACM.

[86] D. Menascé, V. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and J. Wagner Meira. In search of invariants for e-business workloads. In *Proc. of 2nd ACM conference on Electronic Commerce*, pages 56–65, New York, NY, USA, 2000. ACM.

[87] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes. Business-oriented resource management policies for e-commerce servers. *Performance Evaluation*, 42:223–239, 2000.

[88] M. Mitzenmacher. Dynamic models for file sizes and double pareto distributions. *Internet Mathematics*, 1:305–333, 2001.

[89] V. S. Mookerjee and Y. Tan. Analysis of a least recently used cache management policy for web browsers. *Operations Research*, 50(2):345–357, 2002.

[90] M. Motoyama, B. Meeder, K. Levchenko, G. Voelker, and S. Savage. Measuring Online Service Availability Using Twitter. In *Proc. of 2010 Workshop on Online Social Networks*, 2010.

[91] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos. Effective prediction of web-user accesses: A data mining approach. In *Proceedings of the WEBKDD Workshop*, 2001.

[92] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, 26(3):22–36, 1996.

[93] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *ACM Sigplan Notices*, pages 205–216, 1998.

[94] B. Pan. Capturing users' behavior in the national science digital library (nsdl). Technical report, Cornell University Department of Communications, 2003.

[95] K. Park, V. S. Pai, K.-W. Lee, and S. Calo. Securing web service by automatic robot detection. In *Proc. of Annual USENIX Technical Conference*, pages 23–23, 2006.

[96] M. B. Prince, B. M. Dahl, L. Holloway, A. M. Keller, and E. Langheinrich. Understanding how spammers steal your e-mail address: An analysis of the first six months of data from project honey pot. In *CEAS*, 2005.

[97] M. Qureshi, A. Younus, and F. Rojas. Analyzing the web crawler as a feed forward engine for an efficient solution to the search problem in the minimum amount of time through a distributed framework. In *Information Science and Applications (ICISA), 2010 International Conference on*, pages 1 –8, april 2010.

[98] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[99] M. Rabinovich, J. Chase, and S. Gadde. Not all hits are created equal: cooperative proxy caching over a wide-area network. *Computer Networks and ISDN Systems*, 30(22):2253–2259, 1998.

[100] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 111–122, 2003.

[101] C. V. Rijsbergen. *Information Retrieval*. Butterworth, 2nd edition, 1979.

[102] S. Rixner. Memory controller optimizations for web servers. In *Proc. of 37th IEEE Intl. Symposium on Microarchitecture*, 2004.

[103] F. Robinson, A. Apon, D. Brewer, L. Dowdy, D. Hoffman, and B. Lu. Initial starting point analysis for k-means clustering: A case study. In *Proc. of ALAR 2006 Conference on Applied Research in Information Technology*, 2006.

[104] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423, 1948.

[105] W. Shi, Y. Wright, E. Collins, and V. Karamcheti. Workload characterization of a personalized web site and its implications for dynamic content caching. In *Proc. of the 7th International Workshop on Web Caching and Content Distribution*, pages 1–16, 2002.

[106] M. H. Shirali-Shahreza and M. Shirali-Shahreza. An anti-sms-spam using captcha. In *Proc. of 2008 ISECS International Colloquium on Computing, Communication, Control, and Management*, pages 318–321, 2008.

[107] A. Stassopoulou and M. D. Dikaiakos. A probabilistic reasoning approach for discovering web crawler sessions. In *Proc. of the joint 9th Asia-Pacific Web*, pages 265–272, Huang Shan, China, 2007.

[108] A. Stassopoulou and M. D. Dikaiakos. Web robot detection: A probabilistic reasoning approach. *Computer Networks*, 53:265–278, 2009.

[109] Y. Sun, I. Councill, and C. Giles. The ethicality of web crawlers. In *Proc. of IEEE/WIC/ACM Joint Intl. Conference on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 668–675, 31 2010-sept. 3 2010.

[110] Y. Sun and C. L. Giles. Estimating the Web Robot Population. In *Proc. of Intl. World Wide Web Conference*, pages 1189–1190, 2010.

[111] J. A. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[112] P.-N. Tan and V. Kumar. Discovery of Web robot sessions based on their navigational patterns. *Data Mining and Knowledge Discovery*, 6(1):9–35, 2002.

[113] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison-Wesley, 2006.

[114] B. Thompson. Pre-fetching linked content, 2003. US Patent App. 10/746,816.

[115] K. S. Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications.* John Wiley & Sons, Inc., 2nd edition, 2002.

[116] A. M. Turing. Computing machinery and intelligence (1950). *Mind*, 59:433–460, 1989.

[117] `http://planet-lab.org`. Planetlab - an open platform for developing, deploying, and accessing planetary-scale services. Technical report, Princeton University, 2003.

[118] K. V. Vishwanath and A. Vahdat. Realistic and responsive network traffic generation. *SIGCOMM Computer Communication Review*, 36(4):111–122, 2006.

[119] Q. H. Vuong. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica*, 57:307–333, 1989.

[120] C. Wagner, S. Mitter, C. Körner, and M. Strohmaier. When social bots attack: Modeling susceptibility of users in online social networks. In *Proc. of Intl. Conference on World Wide Web*, volume 12, 2012.

[121] A. H. Wang. Detecting spam bots in online social networking sites: a machine learning approach. In *Proc. of 24th IFIP WG 11.3 conference on Data and applications security and privacy*, pages 335–342, 2010.

[122] J. Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.

[123] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level. *IEEE/ACM Transactions on Networking*, 5:71–86, 1997.

[124] J. Y. Yam and T. W. Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1):219–232, 2000.

[125] Q. Yang and H. H. Zhang. Web-log mining for predictive web caching. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):1050–1053, 2003.

[126] S. Ye, G. Lu, and X. Li. Workload-aware Web crawling and server workload detection. In *Proc. Asia-Pacific Advanced Network Research Workshop (APAN'04)*, pages 263–269, 2004.

[127] J. X. Yu, Y. Ou, C. Zhang, and S. Zhang. Identifying interesting customers through web log classification. *IEEE Intelligent Systems*, 20(3):55–59, 2005.

[128] D. Zhong and H. Zhang. Clustering methods for video browsing and annotation. Technical report, In SPIE Conference on Storage and Retrieval for Image and Video Databases, 1997.