NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
Number: NU-EECS-15-03

October, 2015

Measuring Web Servers' Popularity from an Endpoint

Ao-Jan Su and Aleksandar Kuzmanovic

Abstract

Understanding how popular (or not) a given Web service or application is, is an important question whose relevance only continues to grow with the Internet's commercialization. We devise Endpoint Web Monitor (EWM), a novel method capable of measuring a Web server's popularity *directly* from an endpoint. We implement the EWM system and demonstrate that it achieves high measurement accuracy and has a short convergence time which enables it to monitor the popularity trends over short time scales.

Keywords

Web; measurements; popularity;

Measuring Web Servers' Popularity from an Endpoint

Ao-Jan Su Aleksandar Kuzmanovic
Northwestern University, Evanston, IL, USA
{ajsu,akuzma}@cs.northwestern.edu

Abstract

Understanding how popular (or not) a given Web service or application is, is an important question whose relevance only continues to grow with the Internet's commercialization. The state-of-the-art systems (*e.g.*, Alexa, comScore) rely on "crowd-sourcing" methods that attempt to answer this question indirectly, by collecting browsing statistics of a subset of Internet users. In this paper, we take the opposite approach and devise Endpoint Web Monitor (EWM), a novel method capable of measuring a Web server's popularity *directly* from an endpoint.

The key mechanisms unique to EWM are a combination of HTTP `HEAD` requests, TCP pings, and a conservative control mechanism to estimate the number of active connections attached to a Web server. We implement the EWM system and demonstrate that it (i) achieves a high measurement accuracy, (ii) places a negligible computational and bandwidth overhead on the monitored Web servers, and (iii) has a short convergence time which enables it to monitor the popularity trends over short time scales. We deploy EWM on the Internet and demonstrate its practicality in estimating the popularity trends of several Web services (a blogging Web site, eBay auctions, and antivirus software updates) by monitoring accesses to the corresponding server clusters.

1 Introduction

Knowing how many clients are accessing a Web server at any point in time is a useful measure for a number of reasons. In addition to helping with classical network or Web-server traffic engineering and capacity planning tasks, it also reflects the *popularity* of a given service or application hosted at a Web server. A high popularity (or a lack thereof) directly affects the marketing potential of a given application or service. Given that online advertising has become the de facto business model of today's Web, the popularity measure directly affects the cost of advertisements at a given service, which further directly impacts its revenues.

Monitoring the number of client accesses to a Web server is straightforward when one has an administrative access to either server or network logs (*e.g.*, [14, 19, 23]). The hard problem lies in *independently auditing* popularity, *i.e.*, without any administrative privileges. The common, yet inherently inaccurate, approach to this problem, applied by services such as Alexa [2], comScore [8], or Google Trends [11], is "crowd-sourcing": a subset of Internet clients installs free toolbars and other software to collect user browsing statistics in exchange for virus scanning software, Internet data storage, *etc*.

The problem with crowd-sourcing methods is that they are necessarily not comprehensive since they rely only on a subset of end users. Consequently, such methods can provide estimates with *unknown* error bounds. It is thus not a surprise that the corresponding measurement results are often questioned for their accuracy (*e.g.*, [6, 15]). Empirical measurements have shown that such methods (i) can generate striking discrepancies relative to the ground truth data, and (ii) have a fundamental inability to accurately estimate popularity trends [6, 15].

In this paper, we present Endpoint Web Monitor (EWM), a system capable of accurately estimating the number of active connections established to a Web server at any point in time. Unique to our approach is its ability to *directly* and independently estimate the number of active connections to a Web server, without *any* large-scale "crowd-sourcing" client software distribution. Our method incurs a negligible computational and bandwidth overhead, which makes it practical to deploy. Moreover, contrary to other approaches [2, 8, 11], our system is capable of accurately estimating popularity trends over short time scales.

Our approach uses a combination of HTTP `HEAD` requests and TCP pings to decouple network latency and server processing time. To accurately detect the num-

ber of active connections attached to a server, the system injects additional "artificial connections" to the measured Web servers in order to increase the server load over *short time scales*. Similarly to the related networking approach [25], our system utilizes a multiplicative increase multiplicative decrease (MIMD) control system as well as historical information to achieve fast convergence time. It is essential to understand that our system is capable of quickly converging towards the "knee" point of the Web server's accept queue, thus making no impact on regular clients' browsing performance.

We extensively evaluate EWM in our controlled testbed environment and on the Internet with real world Web traffic patterns and Web clients. In particular, we evaluate EWM on a three-tier Web service, which is a stereotype architecture for many Web sites such as social networking, e-commerce and news Web sites. Next, we explore EWM's performance using real-world accesses to a blogging Web site. Finally, we utilize EWM to explore eBay clients' accesses during online auctions.

Our evaluations reveal the necessary trade-off between the measurement accuracy and convergence time. Nonetheless, we show that our system is capable of achieving a desirable point between the two: the average measurement accuracy of 85% and the average convergence time of 13.5 seconds. Most importantly, we demonstrate that EWM's *peak* measurement overhead during the convergence period is negligible, *i.e.*, it is 4.37 kb/s, which makes little to no impact to the daily operations of a Web server. Indeed, our empirically demonstrate that EWM's impact on regular clients' transaction times is negligible, *i.e.*, response times increase on average by about 2% relative to the scenario when EWM is not used. Moreover, we have verified that EWM does not exhaust any computational resources of the monitored Web server, *i.e.*, CPU, memory, or network bandwidth.

We also analyze the scalability properties of our system. We demonstrate that the small measurement overhead induced by EWM enables extraordinary scalability properties. In particular, a conservative analysis shows that a cluster of 315 servers and access bandwidth of 96 Mb/s can effectively monitor as many as 1 *million* Web servers over time scales of 10 minutes.

In addition to monitoring any of a large number of autonomously hosted Web sites, our system is capable of monitoring any autonomously-hosted Web-based services and applications on the Internet. Necessarily, in this paper we focus an several example scenarios to demonstrate the practicality of our approach in the wild. In particular, we monitor the short time-scale popularity trends of well-known antivirus and open-source software vendors by measuring the corresponding server clusters. These experiments clearly demonstrate EWM's ability to directly monitor real-world Web services and measure the Web traffic that they are attracting.

2 Design Goals

	Endpoint	**Client Support**
Own the Web Sites	Web Log Analysis [5, 22], Network Traffic Monitoring [7, 20]	Page Tagging [9, 16, 21], Google Analytics [10]
Independent Measurement	EWM	Alexa [2], comScore [8], Google Trends [11]

Table 1: Web Traffic Measurement Design Space

Table 1 outlines the design space for Web traffic measurement systems. Different methods and associated tools (that we explain in detail below) can be classified in different categories based on two key parameters. The first parameter is the *administrative authority*, *i.e.*, one can (i) own the Web sites and thus can directly access server or network logs; otherwise, one must perform (ii) external independent measurements to estimate the traffic volumes. The second parameter is *client support*, *i.e.*, the question is whether (iii) Web clients' support is required in obtaining measurements, as it is the case with the "crowd-sourcing" example outlined above. Alternatively, if no clients' support is required, the measurement can be obtained (iv) directly from an endpoint. To the best of our knowledge, EWM is the first Web traffic measurement system that requires neither ownership of Web sites nor Web clients' support, *i.e.*, it can perform external measurements independently from an endpoint. The key design goals for EWM are the following:

- High accuracy. It is desirable that the measurement system reports the Web traffic accessing a Web server with a high accuracy. In addition, the accuracy should not be affected by the network distance between the measuring vantage point and the monitored Web server.

- Negligible measurement overhead. The measurement overhead of the measurement system should not impact the daily operations of the monitored Web server. It should neither consume a large network bandwidth nor extensive Web server's compu-

tational resources (*e.g.*, CPU cycles, memory or file I/O).

- Fast convergence time. The average time required to complete a valid estimation should be as small as possible. Not only it can minimize the impact to the observed Web server but also enable the measurement system to monitor the dynamics of a Web server with a finer granularity, *i.e.*, over short time scales.

- Non-intrusive measurement methodology. The measurement system should not attempt to obtain the privilege to enter the monitored system nor to eavesdrop packets from the internal network or the Internet. In addition, the measurement system and its methods should not be misinterpreted as a denial-of-service attack by the monitored system.

- Independent endpoint measurements. The measurement system should be completely independent from the monitored system, *i.e.*, it should rely on external measurements. In addition, the measurement system should be executable from an endpoint, *i.e.*, it should not rely on any large-scale crowd-sourcing efforts.

The first four goals of Web traffic measurement can be easily achieved if one owns a Web site [14,27]. Accurate estimates could then be obtained from an endpoint, by analyzing either server log files or network traces. Table 1's upper left quadrant shows this point in the design space. There are several well-established related approaches that enable webmasters to track the traffic accessing their Web sites.

Web log analysis software, *e.g.*, [5, 22], extracts Web traffic information from the Web servers' access logs. While this approach is certainly accurate, reporting real-time Web traffic from Web logs is not as trivial a task as it may appear. It is often the case that Web servers' log files are stored locally and taken out for offline processing on another server that runs the analysis software. Network traffic monitors, *e.g.*, [7, 20], capture Web traffic to a Web server by monitoring network packets that are transferred toward a server. One advantage of this approach is that Web server's network load can be captured instantaneously. Still, in practice, not many webmasters have the privilege to deploy a sniffer in the network to collect packets because such sniffers may sometimes violate a company's privacy policy.

While the above approaches that rely on server or network administrative access privileges certainly have their advantages, *i.e.*, they provide accurate results, the key challenge lies in developing a method to independently audit Web servers' popularity. Alternatively, an obscure Web site can report that it attracts millions of users daily; the question is how to independently validate such claims.

We next move to the second quadrant in the design space shown in Table 1, *i.e.*, the one where both administrative privileges and client support are available. With Web clients' support, there are additional opportunities for a webmaster to collect information about the Web traffic and clients. Page Tagging [16, 21], a method also known as a "Web Bug" [9], involves placing a piece of javascript code on each Web page of a Web site. Every time a tagged page is opened by a Web client's browser, the script is processed and the visitors' information (such as cookies, browser and hardware information) is collected. While this approach is capable of collecting much more detailed information about end users, it still depends on the willingness of end users to share such information with Web sites. If clients disable javascripts, which they typically do for security reasons, this approach is ineffective. In addition, page tagging cannot track "non-pages" including media files, pdf and zipped files *etc*.

Another approach that we place in the second quadrant of Table 1 is the one applied by Google Analytics [10]. A webmaster that subscribes to this service can gather information about clients that are redirected from the Google search engine to the Web site administered by the webmaster. Client support is needed in the sense that clients must actively click links on the search engine in order to be accounted using this method. Necessarily, information obtained in this way is limited to click-through requests that come to the site. Hence, such traffic reports are limited to the search engines' users and cannot reflect the total Web traffic accessing the Web server.

Common for the approaches in the second quadrant is the additional information about clients collected by Web sites. Still, such information is available only to the Web sites, not the general public. Moreover, information collected in this approach is not comprehensive because it either requires explicit clients' support [16, 21] or depends solely on click-through requests [10].

The demand for systems capable of independently auditing Web site's popularity is currently fulfilled by the ones shown in the fourth quadrant of Table 1. In particular, these are Alexa [2], comScore [8], and Google Trends [11]. As discussed above, such systems require a large-scale deployment of client software that collects user browsing statistics. They then extract user browsing statistics from a subset of end users that install such software, and project such results to all Internet users. It is not a surprise that such data can be biased to a subset of Internet users and its accuracy is often being questioned [6, 15].

Our approach is to explore the opposite (third) quad-

rant in Table 1. Is it possible to achieve the five goals of Web traffic measurement without owning the Web sites and without any clients' support? In doing so, we design EWM to probe a Web server *directly*, *i.e.*, without relying on necessarily incomprehensive statistics collected by a subset of end users, and report the popularity of a Web server in real time. We next present the EWM details below.

3 Methodology

In this section, we present the EWM design and describe its elements. We first provide the necessary background and explain how a Web server works. Then, we present the EWM system architecture and its probing and control mechanisms that enable us to measure the number of active connections to a Web server. Finally, we analyze EWM's measurement overhead.

3.1 Dynamics of a Web Server

Here, we provide the necessary background relevant for this paper on how a Web server works. When a Web client tries to request service from a Web server, it first establishes a TCP connection to the Web server. The connection is then queued in the Web server's listen socket's accept queue to wait for an available Web server subprocess or thread to process the request. If the Web server has a spare process waiting to serve the request, the request is assigned the spare process and it is executed immediately. When all the processes in the service pool are occupied to serve the existing connections, the new coming request has to wait in the Web server's accept queue for a Web server's process to free up. In this case, the service time of the request will increase with the number of connections waiting in the accept queue.

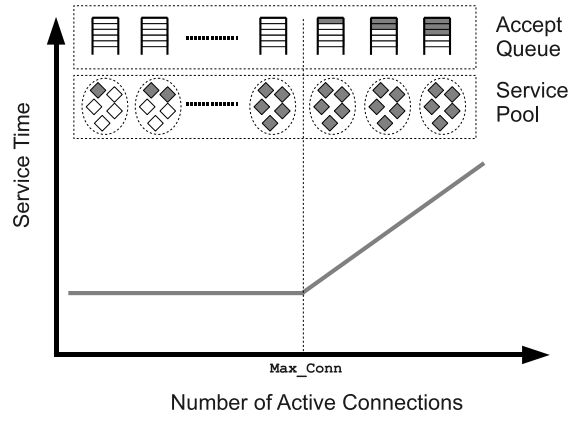

Figure 1: Server's Service Time vs. no. of Active Connections

Figure 1 shows the relationship between the service time and the number of active connections to the Web server. When the number of active connections does not exceed the limitation of maximum connections (`Max_Conn`),[1] the request is processed immediately, *i.e.*, it is not queued in the accept queue. Hence, the entire request service time equals the request processing time. For the same type of requests that we use in our probing methodology (to be explained in detail below), the service time is small and identical when the system operates below the "knee" point shown in the figure. When the number of active connections increases and exceeds the Web server's limitation of maximum concurrent connections, the Web requests start to accumulate in the listen socket's accept queue and the request service time starts to increase due to the queuing delays.

The key idea standing behind the EWM method is to estimate the number of active connections attached to a Web server by *gently* shifting (over short time scales) the Web server's operating point close to the knee point shown in Figure 1. Such an approach enables accurate estimates of the number of active connections to the Web server, as we demonstrate below.

3.2 System Architecture

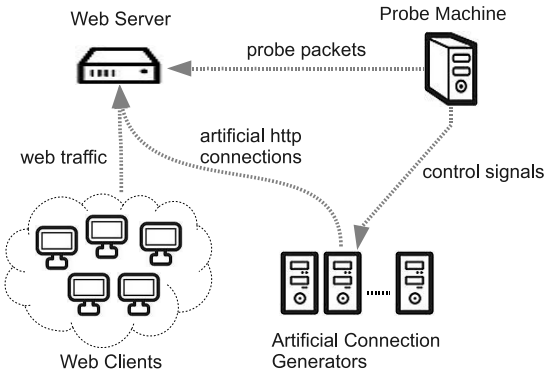

Figure 2: System Architecture

Figure 2 depicts the EWM system architecture. It consists of two main components: (i) a probing system that sends probing packets to the monitored Web server, and (ii) an artificial connection generator, a cluster of machines used to shift the Web server's operating point to the "knee" service point shown in Figure 1. The probing machine sends a combination of TCP and HTTP probe packets to the monitored Web server. These probe packets measure the Web server's service time and monitor

[1] A Web server sets a limit for the number of connections (`Max_Conn`) that it can process concurrently (*e.g.,* `MaxClients` in Apache Web server's configuration).

the queue length in the Web server's accept queue. In addition, the probe machine sends control messages to the artificial connection generator to control the number of artificial connections sent to the monitored Web server. We explain the probing and the artificial generation mechanisms in more detail below.

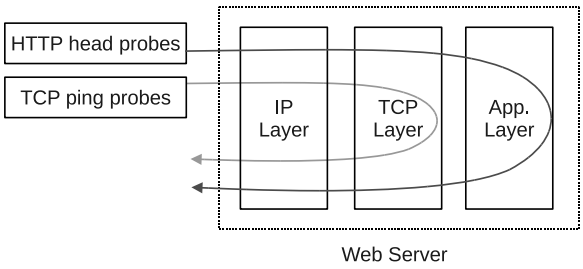

Figure 3: EWM Probes

Figure 3 depicts how the EWM probes work. The TCP ping probe (a TCP ACK / RST pair, explained in detail below) travels from the probe machine to the monitored Web server's TCP layer. EWM measures the round trip time of this probe and records it as the network latency from the probe machine to the Web server. The HTTP probe sends an HTTP `HEAD` request from the probe machine to the Web server. The round trip time of this probe equals the network latency plus the server's service time. EWM estimates the Web server's service time by subtracting the TCP ping probe latency from the HTTP probe's latency. As explained above, when the server's operating point is below the knee point, the measured service time will not be affected by the accept queue delay. Consequently, the distance between the two probes is minimal.

EWM expects to detect when the service pool is full, *i.e.*, when the service time starts to increase due to request accumulation in the accept queue. The artificial connection generator is used to shift the system to this operating point over short time scales. It generates *unfinished* (or incomplete) HTTP `HEAD` requests to the monitored Web server (*e.g.*, an HTTP header without the last carriage return and the line feed symbols). The Web server will keep the connection alive because it has to wait for the complete header. The unfinished `HTTP` requests place a negligible demand for a server's computational resources. However, they hold service slots for the monitored Web server. In addition, the Web server keeps the connection alive until the artificial generator disconnects or the Web servers' connection timeout expires (typically 300 seconds). In practice, the EWM system keeps artificial connections alive over time scales of several seconds, which is the time required for our algorithm to converge.

Once the algorithm converges (we explain the exact condition in Section 3.3.2 below), the total number of connections attached to the Web server is approximately `Max_Conn`. Hence, we compute the number of active connections (generated by real Web clients shown in Figure 2) by subtracting the number of artificial connections established by the connection generator from the `Max_Conn` number. EWM estimates the `Max_Conn` parameter by monitoring the maximum number of artificial connections that can connect to the Web server over longer time scales, *i.e.*, 24 hours. We also keep the `Max_Conn` parameter as part of the history information for a Web server. Our experiments on the Internet show that the `Max_Conn` is typically set by default to 256. Nonetheless, we have observed other values (*e.g.*, 150) as well.

3.3 EWM Design

3.3.1 Probe Packets

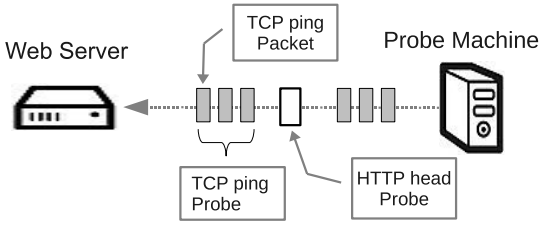

Figure 4: EWM Probing Epoch

Figure 4 illustrates two EWM probing methods, TCP ping probes and HTTP `HEAD` probes. Next, we explain these probing types in detail.

TCP ping packets. A probing machine sends a TCP ACK packet with an arbitrary ACK number to the monitored Web server. Since the Web server cannot recognize this TCP ACK packet, it responds with a TCP RST packet to the sender. We use TCP ACK packets instead of UDP or ICMP probes for two reasons. First, many routers and firewalls drop UDP or ICMP probes, or treat them with lower priority than TCP packets, which would impact our system's accuracy. Second, TCP ACK probes raise fewer security alarms than other probes [34]. EWM measures the round trip time (RTT) of the corresponding ACK and RST packets and estimates the network latency from the probe machine to the monitored Web server.

HTTP `HEAD` probes. An HTTP `HEAD` probe consists of a HTTP `HEAD` request that is sent to the monitored Web server. The HTTP `HEAD` request asks for the HTTP header of a Web page and it is often used to test recent modifications of the Web page. It is a lightweighted HTTP message because the protocol transfers only an HTTP header of an HTTP object (without the HTTP body). Hence, the overhead of this message is

small for all file types (*e.g.*, music, zip, pdf, *etc*). The latency of this probe accumulates the network latency and the server's service time. When the server operates below the knee point, the service time is minimal and the latency of the HTTP probe will be very close to the latency of a TCP ping probe. On the other hand, when the queue is building up in the Web server's accept queue, the HTTP `HEAD` probe's service time will accumulate accordingly.

EWM probes. As illustrated in Figure 4, EWM sends three back-to-back TCP ping packets (one TCP ping probe) in front of and behind an HTTP `HEAD` probe. This is done in order to accurately detect potential network congestion events (reflected in an increased RTT variance) that may interfere with a Web server's service time measurements, as we explain below.

In particular, for each round of the measurement, the probe machine sends three *probes* to the monitored Web server including: one TCP ping probe, one HTTP `HEAD` probe, and another TCP ping probe, respectively. The three probes are sent back-to-back to the monitored Web server and EWM measures the response time of each request. The two TCP probes estimate the network latency from the start of the measurement to the end of the measurement. If network congestion happens during our measurement, the difference of the two sets of TCP pings will be significant and EWM will simply disregard this round because it cannot tell how much of the delay is contributed from the Web server's service time. If there is no network congestion variance, the Web server's service time can be accurately estimated by subtracting the RTT of the TCP ping probe from the turn around time of the HTTP `HEAD` request.

3.3.2 Artificial Connections Control

EWM generates artificial connections to the monitored Web server in an attempt to shift the system into the knee operating point shown in Figure 1. Here, we explain the control mechanisms that EWM uses to reach this goal.

In each measurement round, EWM applies a conservative multiplicative increase and multiplicative decrease (MIMD) control mechanism to control the number of artificial connections to a Web server. Denote by *OldConn* the number of artificial connections established to the Web server in the previous measurement round. Next, denote by *m* the scaling parameter. Further, denote by *Rand* a random integer between -5 and 5 excluding 0. Then, the number of connections that will be established to the server in the current measurement round, *NewConn*, becomes

$$NewConn = m * OldConn + Rand. \quad (1)$$

The *m* parameter is set separately in multiplicative increase and decrease phase. We set $m = 1.5$ when in MI phase and $m = 0.75$ when in MD phase. An extensive experimental evaluation (that we omit here for space constrains) has shown that this provides desirable convergence properties (details below). Likewise, our experiments show that the *Rand* parameter effectively helps avoid synchronization effects, also common for network controlled systems [33].

Convergence condition. Here, we explain the condition we use to stop the control procedure and disconnect all artificial connections. The key trade-off that we face here is the one between the convergence time and the accuracy of our estimates. We set parameters with the goal of achieving short convergence times, and reasonable accuracy. In particular, denote by C_d the maximum number of artificial connections for which we measure *no* queuing delay in the server's accept queue. Next, denote by C_i the minimum number of artificial connections for which we *do* measure queuing delay in the server's accept queue. Necessarily, $C_i > C_d$. Then, we consider that the system has converged once the following condition is achieved

$$\frac{C_i - C_d}{MaxConn} < 0.15. \quad (2)$$

Thus, when we bound the range in which the number of artificial connections reside around the knee point to within 15% relative to the maximum number of connections, we converge. Finally, we estimate the number of active connections as $MaxConn - (C_i + C_d)/2$. In this way, we necessarily introduce error, yet we reasonably bound it. Our experiments later in the paper confirm that this is indeed the case. At the same time, this approach enables fast convergence, which is an important goal we were striving for.

3.3.3 History Information

Using historical information to set the initial probing rate is a well-known concept in computer networking [25]. We apply this concept in the EWM design as follows. First, we keep the *history information* about the number of artificial connections needed to trigger the knee operating point. When the history information is available, we use it to set the initial number of artificial connections. Second, EWM also caches the information about the *MaxConn* parameter for a Web server. Given that this parameter is unlikely to change over long time scales, caching it is certainly beneficial. We experimentally evaluate the benefits of using the history information below.

3.4 Measurement Overhead

Here, we analyze the measurement overhead incurred by our system. As shown in Figure 3, in each measurement round we send one TCP ping probe, followed by an HTTP `HEAD` probe, followed by another TCP ping probe. For TCP ping probes, the overhead for each probe is approximately 0.32 kBytes. For each HTTP `HEAD` probe, the overhead is 1 kByte. Therefore, the measurement overhead for each measurement round is approximately 1.64 kBytes. Given that EWM's average convergence time is approximately 4.5 rounds (as we will demonstrate later in the paper), the measurement overhead for the 4.5 rounds is 7.38 kBytes. Given that each measurement round is upper bounded by 3 seconds, the duration of a typical convergence epoch is 13.5 seconds. Thus, the bandwidth overhead placed on the monitored server is 4.37 kb/s during the measurement epoch. Note that the 4.37 kb/s overhead would not increase if EWM needs more rounds to converge because EWM paces itself for 3 seconds per round and the overhead of the probe packets is fixed.

In general, EWM can schedule probing epochs over arbitrary time intervals ($\geq$ 13.5 seconds), depending on the time scales at which an endpoint desires to estimate the popularity trends. For example, in scenarios when EWM is measuring a large Web server infrastructure (*e.g.*, mirror sites of a Web service, Section 5), EWM can schedule measurement intervals over longer time scales (*e.g.*, 5 minutes) in a round robin fashion. In this case, the average measurement overhead is necessarily lower. Therefore, the EWM's overhead will not disturb the monitored Web server's daily operations and the network overhead is negligible.

4 Evaluation

In this section, we experimentally evaluate EWM. Common for all scenarios in this section is the existence of ground-truth information, *i.e.*, real server logs, which enables us to systematically evaluate EWM's accuracy.

4.1 A Three-Tier Web Service Experiment

Here, we evaluate EWM in a three-tier Web service testbed. The three-tier Web service is a stereotype architecture for many Web sites such as social networking, e-commerce and news Web sites. It consists of a Web server, an application server and a database, as illustrated in Figure 5.

We deploy an eBay-like auction service, RUBiS [17], on our three-tier testbed and generate different workload scenarios using RUBiS's workload generator. The workload generator spawns a different number of concurrent

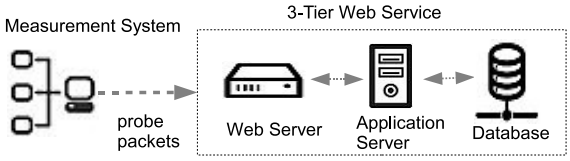

Figure 5: Three-Tier Web Service

Web clients with a series of transactions such as browsing, search, and bidding. The transactions produce dynamic arrival and service times to simulate the realistic Web traffic. In the experiments, we initiate a different number of Web clients ranging from 32 to 512. Web clients in the experiment can disconnect and pause for a certain amount of time, thus simulating the "think" process common for real Web clients. We deploy EWM to monitor the active connections to the Web server in the three-tier Web service testbed and evaluate its accuracy by comparing our estimations to the Web server's log files.

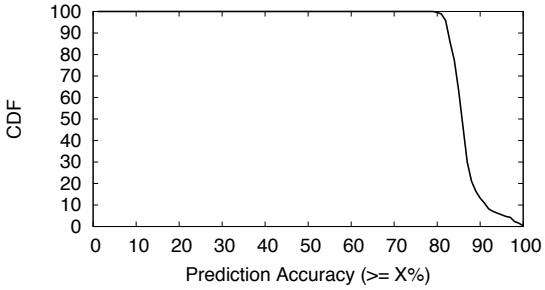

Figure 6: EWM's Accuracy in the Testbed Experiment

Figure 6 shows EWM's accuracy for our testbed experiments under different workload. In this figure, the x-axis represents the prediction accuracy of the estimated number of active connections relative to the ground-truth data, *i.e.*, connections recorded in the Web server's log. The y-axis represents the cumulative distribution function of accuracy in all experiments. The figure shows that in all of the workload scenarios, the EWM accuracy is greater than 80% and the average accuracy is 85%. This result is a direct consequence of our design decision outlined by Eq. (2) above. In particular, our goal is to reasonably bound the estimation error while achieving fast convergence. Below, we evaluate EWM's convergence time in addition to accuracy in real-world Web traffic scenarios.

4.2 Monitoring Blogging Web Patterns

Here, we evaluate EWM by replaying a real-world one-hour-long Web server log obtained from a blogging Web site. According to Alexa, this blogging Web site ranks approximately at top 8,500 Web sites in the world and allegedly attracts non-negligible Web traffic every day. This experiment helps us understand EWM's accuracy in measuring a popular real-world Web site. The log we obtained contains timestamps of Web clients that send requests as well as the duration of the requests. The requests correspond to a combination of accesses to files of different sizes (*e.g.*, html, image, and music) and different types of requests (*e.g.*, search, browsing, and post requests).

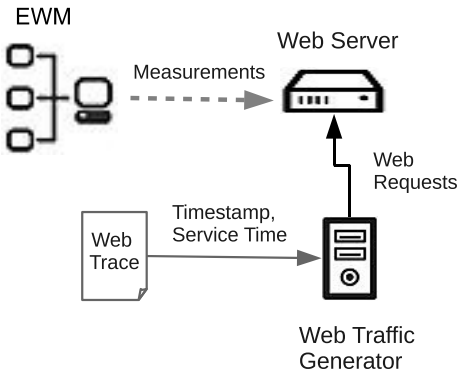

Figure 7: Replay Web Server Log Experiment Setup

Figure 7 shows the experimental setup. We set up a Web server in our testbed to replay this Web log and monitor the active connection to this Web server using EWM. To replay a single request from the log, we initiate a Web client to the Web server at the given timestamp in the log and send the processing time (*e.g.*, *n* seconds) as a parameter in the URL to the Web server. Once the Web server receives the HTTP request, it parses the URL and keeps the Web client for the amount of time indicated in the processing time (*i.e.*, *n* seconds in our example). After the processing time expires, the Web server sends back a HTTP 200 OK reply to the Web client. Similarly, we repeat the process until all the Web traffic log is replayed. Simultaneously, EWM probes the Web server every 30 seconds to monitor the number of active connections to the Web server. We record the results and compare with the Web server's log file after the experiment is completed.

Figure 8 shows the results of the experiment in which we replay the one-hour-long Web server log. The x-axis shows the timestamps of the experiment in seconds and the y-axis shows the number of active connections. The figure demonstrates that the recorded active connections from the Web log are closely approximated by

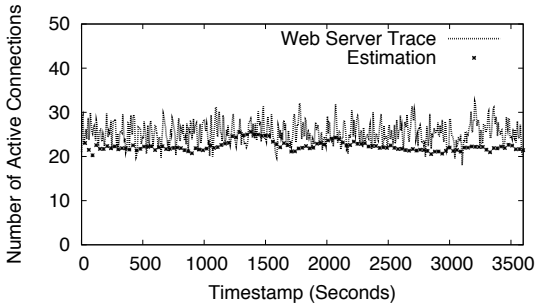

Figure 8: Monitoring Blogging Web Patterns

EWM's estimations. Though EWM tends to underestimate the number of active connections in this case because it misses some very short-lived Web requests, its average accuracy is 87.5%.

Next, we study the advantage of using history information. We compare the number of rounds required for convergence in two scenarios: with history information and without history information. When history information is used, EWM utilizes the number of artificial connection observed in the previous convergence round to jump start the new measurement. When history information is not used, EWM selects a random number between 32 and *MaxConn* as the initial number for artificial connections.

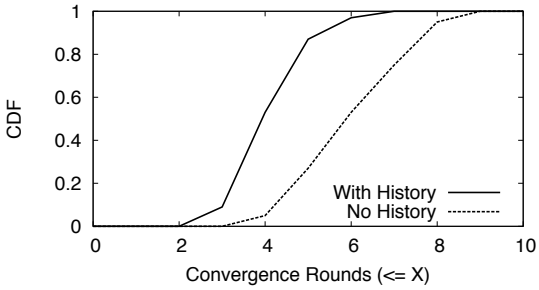

Figure 9: Advantage of History Information

Figure 9 plots the CDF of the number of rounds needed for the algorithm to converge in the above experiment. The figure shows that when history information is used, more than 50% of measurements converge within 4 rounds, and the average number of rounds is 4.53. Given that the theoretical minimum for the algorithm to converge is 2 rounds (*i.e.*, at least two estimates

are needed to compute the convergence condition, Eq. (2)), this is indeed a short convergence time. The figure also shows that when history information is not used, the average number of rounds needed for the system to converge increases by 50% approximately, *i.e.*, to 6.45. The experiment clearly shows the power of using the history information to shorten EWM's convergence time.

4.3 Monitoring eBay Clients

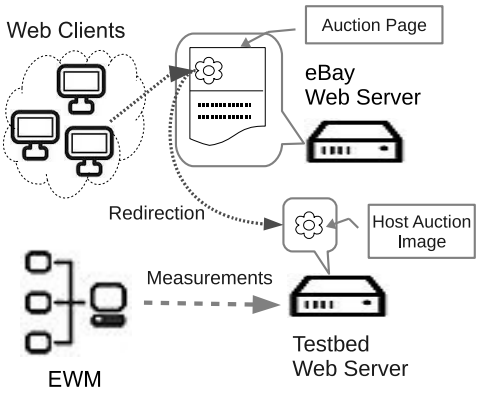

Figure 10: eBay Experiment Setup

Here, we extend the EWM evaluation to measure *real Web clients* in *real time*. In particular, we posted 10 auctions with very close auction end times on eBay. In addition, we hosted images in the auction pages on our testing Web server. As shown in Figure 10, when a bidder is accessing (or reloading) the auction page, the bidder will be redirected to our testbed Web server to download the image. As such, we can measure the Web traffic accessing those auction pages indirectly by measuring accesses to the Web server in our testbed using EWM. Of course, we can measure eBay's servers directly. The benefit of the setup shown in Figure 10 is that we have the ground-truth information from our Web server, while at the same time we measure real Web clients in real time.

Figure 11 depicts the number of active connections from EWM's measurement results when the auction is closing. The x-axis is the timestamp relative to the auctions closing time (*i.e.,* 0 is auctions' end time). Our measurement results confirm the last-minute bidding behavior described in several studies (*e.g.*, [41, 42]) that online bidders tend to bid at the last minute before auctions end. When the auctions are closing, EWM monitors a peak of Web traffic accessing the Web server that hosts the auction images. After the auctions end, the Web traffic drops significantly because bidders leave after the auction winner is declared. The key point is that EWM is capable of effectively estimating non-stationary popularity trends over short time scales and with high accuracy.

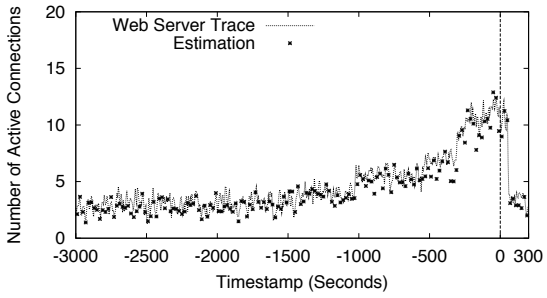

Figure 11: Monitoring Web Traffic from Ebay Clients

5 Case Studies

5.1 Monitoring Antivirus Update Server Clusters

With the prevalence of endless computer viruses and malware, antivirus programs have become an indispensable software in every computer. While some users choose their antivirus software by reputation or cost, others choose the software based on its popularity. However, the popularity of an antivirus software cannot be easily measured. In particular, some Web sites provide the count of software downloads, but such numbers cannot be independently verified. Moreover, even if accurate, such numbers can be far from the real number of *"live users"*. For example, a user can download a piece of software but never install it or stop using it (*e.g.*, replace with the competitors' antivirus software).

Here, we provide an alternative method to evaluate the popularity of antivirus software. We utilize EWM to monitor the virus definition update Web servers of antivirus software. Essentially, an antivirus software installed on a computer will connect to the update Web server to update its virus definitions periodically. Therefore, the volume of Web traffic to the update server can indirectly show the popularity of the antivirus software. In this case study, we focus on two popular *free* antivirus software vendors, `Avira` [4] and `Avast` [3].

To monitor the antivirus software update servers, we first obtain the hostname of the update Web server by sniffing the packets sent out from our computer to the server. Next, we perform DNS lookups for the hostnames we obtain. For `Avira`, the hostname (`personal.avira-update.com`) resolves to 30 unique IP addresses that are associated with 30 Web servers. For `Avast`, we obtain 50 hostnames (*e.g.,* `download824.avast.com`) and each hostname is associated with one unique IP address.

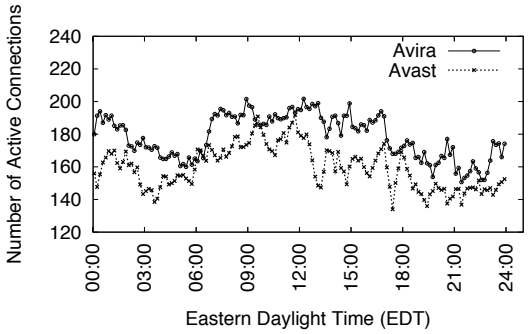

Figure 12: Monitoring Antivirus Software Update Servers

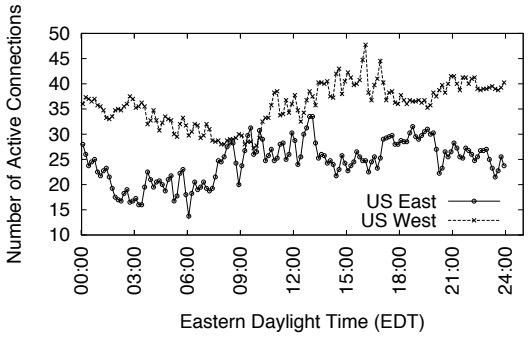

Figure 13: Monitoring SourceForge Mirror Sites

Figure 12 shows the *aggregated* number of active connections from the update servers of the two antivirus software vendors. The figure shows that the two software vendors have comparable popularity with `Avira's` update Web servers attracting slightly more Web traffic. This is despite the fact that `Avira` is hosted at a smaller number of Web servers (*i.e.*, 30) relative to `Avast` (*i.e.*, 50). In addition, the figure also shows the time of day effect. For example, both server clusters have the lowest traffic between 03:00 and 06:00.

We note that our results in this particular case are relative in the sense that they simply show the aggregate Web traffic to the corresponding server clusters. Indeed, how often an antivirus software updates its database can impact the Web traffic to its update servers. In that sense, the comparison of the two software vendors shown in Figure 12 may not be fair. To fairly compare the popularity, we can estimate the update frequency for different systems by simply monitoring software behavior at an endpoint. We do not pursue this avenue here because it diverts from our key point: that EWM provides a means to directly monitor Web services and measure the Web traffic that they are attracting. In addition, monitoring these Web servers over long time scales can reveal a software popularity trend, *i.e.*, if it is attracting more users or getting unpopular.

5.2 Monitoring SourceForge Mirror Sites

SourceForge is a Web-based source code repository that hosts many of the most popular open source software projects (*e.g.,* Ghostscript, BitTorrent and Wireshark). SourceForge depends on its 31 mirror servers located globally to distribute Web traffic generated by software downloads from its Web site. In this case study, we deploy EWM to monitor two of the SourceForge's mirror servers and analyze the results.

Figure 13 shows the number of active connections to mirror servers located at US East Coast (New York) and West Coast (Seattle). The first insight from the figure is that the server on the West Coast often attracts twice as much users than the one at the East Coast. Given that SourceForge redirects Web clients to its closest mirror sites according to their IP addresses, we can say that our measurements to some extent "confirm" the stereotype that people on the West Coast are more "techy" than on the East Coast. Next, the figure also shows a time of day effect of the Web traffic to these mirror servers. We also observe the time difference between the two mirror sites. For the US East mirror server, its traffic starts to increase around 6:05 Eastern Daylight Time (EDT) and the US West mirror server's Web traffic starts to increase around 9:45 EDT which is 6:45 Pacific Daylight Time (PDT). Internet users can use the monitoring results provided by EWM as one of the important inputs (in addition to RTT and bandwidth measurements) to select a mirror server that has less traffic to avoid busy or overloaded mirror servers.

6 Discussion and Additional Evaluation

Active connections vs. active clients. Modern Web browsers can accelerate the loading of a Web page by initiating multiple HTTP connections to a Web server and downloading multiple HTTP objects concurrently [29]. As such, it is possible that one Web client has multiple active HTTP connections to the same Web server. At a high level, this behavior does not conflict with our goal of measuring the popularity of a Web server. Indeed, more active clients generate more active connections. While establishing the number of active clients based on the number of active connections to a Web server is beyond the scope of our work here, we note that such a relationship is possible to establish.

In particular, we have already developed methods to effectively and scalably profile arbitrary Web sites by accounting for the *size* of Web objects hosted by a server, their cacheability and locality [39], all of which affect the relationship between the number of active connections and active clients. For example, for Web servers that host (i) short text files vs. (ii) large pictures and videos, the same number of measured active connections correspond to a larger number of active users in the text server scenario relative to the corresponding pictures and videos case. Thus, based on the characteristics of the content hosted at a Web site, and based on understanding of how modern Web browsers (all disable pipelining and enable caching [39]) utilize multiple connections,[2] it is possible to develop models to compute the most likely number (and a distribution) of active clients based on the number of active connections. We are currently developing such models.

The impact of load balancers. Large Web sites that host a cluster of Web servers may deploy a load balancer to manage a set of Web servers for reliability and load balancing purposes [13]. EWM can monitor the Web server cluster as a single Web server and measure the aggregated active connections to the cluster. Given that the load balancer uniformly distributes the Web requests to its associated Web servers [26,30], EWM works properly in this scenario. We evaluated this issue in our testbed using a load balancer implemented as a software Web proxy - HAProxy [12] and two Web servers. We omit the results here since they are well expected. Monitoring larger-scale clusters (beyond our current setup) certainly requires more measurement resources. We discuss EWM's scalability properties below.

DoS attack concerns. The amount of artificial HTTP connections generated by EWM may be misinterpreted as a denial-of-service attack by the monitored system. There are several issues with respect to this concern. First, EWM has short convergence time and low network overhead that should not be misinterpreted as DoS attacks. Our experiments from Section 5, in which we monitored production Web services on the Internet over long time scales without any obstacles, confirm that this is the case. Moreover, flash crowds are a well known phenomenon for Web services [35, 36]. Any anti-DoS scheme that interprets EWM's measurement as DoS attacks will necessarily reject connections during flash crowds, which is certainly an undesirable behavior.

EWM's Impact on a Web Server. To explore EWM's impact on a Web server, we study EWM's impact on transaction times experienced by regular Web clients. We conduct experiments using the experimental setup from Section 4.2, depicted in Figure 7 above. We compare the transaction time of Web requests with and without EWM measuring the Web server. During EWM's measuring epoch, the Web clients only experience queueing delays when the total number of connections generated by EWM and Web clients exceeds the Web server's *MaxConn* (in the MI phase). This typically happens only in a *single* round during the measurement epoch. Otherwise, EWM does not impose additional delays to the Web requests.

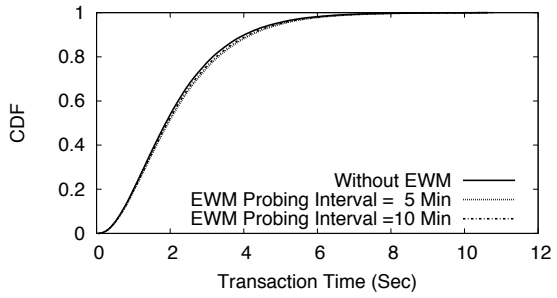

Figure 14: EWM's Impact on Transaction Times

Figure 14 plots the CDF of transaction times perceived by the Web clients. The figure shows that without EWM the average transaction time is 2.19 seconds. With EWM's measurements, the average transaction time slightly increases to 2.28 seconds when EWM's probing interval is 5 minutes. Moreover, when EWM's probing interval is 10 minutes, the average transaction time increases to 2.24 seconds relative to 2.19 seconds when no EWM measurements are conducted. We conclude that EWM's impact on regular clients' transaction times is negligible. Moreover, we have verified that EWM does not exhaust any computational resources of the monitored Web server (*i.e.*, CPU, memory, or network bandwidth) by closely monitoring our testbed's machines.

Synchronized measurements issue. If multiple entities are deploying EWM to measure the same Web server concurrently, the artificial connections generated by one EWM instance will be counted as Web traffic by another one. A similar problem also exists in related computer networking scenarios, *e.g.*, [25, 31], where probes generated by one endpoint can be misinterpreted as the real background traffic by other endpoints.

There are fundamental differences between EWM on one hand, and such networking scenarios on the other. Contrary to related networking scenarios in which it is assumed that each and every endpoint massively and regularly probe the Internet, this is not the case with EWM. We envision that only one (or several at most) entities

[2]Somewhat outdated studies on this topic do exist, *e.g.*, [32, 38].

will adopt EWM to enable popularity auditing services to the public. Any individual entity can appropriately schedule probes to avoid synchronization. Any synchronization effects among potential different entities are unlikely to happen. Indeed, as long as the intervals inbetween two consecutive measurements are reasonably spaced (*e.g.*, 10 minutes), the synchronization is unlikely given the short EWM's convergence times. The synchronized measurement issues can be further alleviated by randomizing the intervals between measurements.

Shared-hosting and Content Distribution Network (CDN) services. In shared-hosting Web server scenarios, *i.e.*, when multiple sites are co-hosted on a single Web server, EWM is able to measure the aggregated Web traffic to the server, not to individual Web sites. While the traffic to such individual Web sites is small by design, we are capable of providing accurate upper bounds for traffic to such Web sites.

Some Web sites use CDNs to replicate their content to many locations on the Internet, thus making it closer to end users. The question is if and how EWM can measure the popularity of such Web sites. To answer this question, we proceed as follows. We take the list of the top 1 million most popular Web sites from Alexa, and search for the Web sites hosted on Akamai [1], the largest CDN provider. We find 6,415 Akamai-hosted Web sites among the top 1 million, *i.e.*, 0.64% of the top 1 million Web sites are hosted on Akamai. Next, we analyze a random subset of such Web sites. We find that despite the fact that the most "heavy" content (*e.g.*, pictures, videos *etc*) is indeed served by Akamai, in the vast majority of cases we find that pieces of content (*e.g.*, typically text) is still served exclusively from origin, non-CDN, servers. EWM can accurately monitor such origin Web servers thus revealing the popularity of the corresponding sites. Moreover, such measurements typically require moderate resources, as we explain in detail below.

EWM's scalability. Consider a scenario in which EWM needs to monitor 1,000 Web servers concurrently during a single convergence period that lasts 13.5 seconds on average. The measurement overhead thus equals to 1,000 * 4.37 kb/s = 4.27 Mb/s. This network bandwidth is certainly within the range that most servers connect to the Internet. Therefore, a single probe machine with multiple probing processes is sufficient. As for the artificial connection generators, assume conservatively that measuring each Web server requires 256 artificial connections, which corresponds to the default maximum number of concurrent connections for a Web server. Then, the total number of artificial connections required for 1,000 servers is 256,000. Given that a typical Linux server can generate 20,000 concurrent sockets [18], it follows that 12.8 artificial generators is enough to handle the task. Note that once an artificial connection is established to the Web server, no packets will be transferred until the connection is closed. Therefore, network bandwidth is not a problem for artificial generators.

In sum, EWM measurement system with a total of approximately 14 servers (1 + 12.8) can monitor 1,000 Web servers in the Internet during a *single* convergence epoch. Considering a realistic scenario in which the inter-measurement interval is set to 10 minutes, and that each convergence interval lasts for 13.5 seconds on average, it follows that 14 servers can concurrently measure 1,000*(10*60/13.5) = 44,444 Web servers. Extrapolating to larger numbers, it follows that a measurement cluster of approximately 315 servers, and aggregate bandwidth of 96 Mb/s can effectively monitor as much as 1,000,000 Web servers concurrently.

6.1 "What If" Scenarios

Manipulating the probing system? Here, we evaluate hypothetical scenarios in which a Web server administrator would intentionally prolong HTTP probes. Indeed, EWM is a delay-based measuring system that is sensitive to irregular increase of Web server's service time. After EWM is widely deployed, it is possible that Web server administrators would want to manipulate EWM's measurement data to artificially increase the popularity of their services. This can be done by uniformly or randomly prolonging HTTP probes sent by EWM. Note that uniformly prolonged HTTP probed can easily be detected and calibrated by EWM using statistical measures.

To detect and filter the intentional randomly prolonged HTTP probes, EWM can deploy multiple probing machines to "audit" the probe machine's measurement results. The monitoring (main) probing machine sends probes and controls the artificial generators as explained above. The auditing probe machines also probe to the monitored Web server at a random time within EWM's measurement epoch to cross-validate the HTTP probe measurement results from the main probe machine. The auditing probe machines should request *random* Web objects from the Web site to avoid detection. Hence, EWM can prune unreliable measurement records from its data set if there are mismatches between the monitoring and auditing probe machines.

Dependence on TCP ping probes? In its current implementation, EWM uses TCP ACK and RST pairs to estimate network-level RTTs. While most servers support such probes [34], one might be concerned that disabling this feature can affect EWM. This is not the case. EWM can use regular (TCP SYN - SYN ACK) pairs to estimate RTT. While such a probe temporarily allocates server resources, the resources can be immediately released by promptly reseting the connection. In summary, EWM is independent from TCP ping probes.

Dependence on HTTP `HEAD` probes? Here, we discuss another hypothetical scenario in which the HTTP `HEAD` protocol is disabled by the Web server's administrator. Web proxies and browsers rely on HTTP `HEAD` protocol to determine the freshness of a Web object. If a Web server administrator disables the HTTP `HEAD` protocol on the Web server, the browsers and proxies will have no choice but to download the whole HTTP objects to check the freshness of the corresponding HTTP objects in their cache. Further, some proxies and browsers may not even work properly because they assume that the HTTP `HEAD` protocol is supported by the Web servers. In addition, when the HTTP `HEAD` protocol is disabled, the network usage of these Web servers will necessarily grow because Web clients have to transfer the entire Web objects. The increased network bandwidth will accordingly cost revenue to the Web administrator.

If the HTTP `HEAD` protocol is disabled (or if HTTP `HEAD` packets are intentionally delayed to manipulate measurements), EWM can adopt an alternative method to send its HTTP probes. EWM can use HTTP `GET` to transfer one of a large number of small and frequently accessed Web object (*e.g.*, the "favorite icon" image of a Web site).[3] Small and frequently accessed Web objects are likely to be cached in the Web server's memory, hence little processing time and computation resources are required for the Web server to transfer the object. In addition, downloading the small HTTP objects will not considerably increase the measurement overhead for EWM and we do not foresee any impact on EWM's accuracy.

As for artificial connections, similarly, the HTTP `GET` protocol can be used in place of the HTTP `HEAD` protocol. The artificial generators can send an HTTP `GET` request with unfinished HTTP header to generate an artificial connection. Furthermore, if a Web server sets a very short connection expiration time that may disconnect the generated artificial connections before EWM converges, the artificial generators can simply connect to the Web server and fetch a series of small HTTP objects during EWM's measurement epoch.

Summary. Using regular TCP packets and Web requests to random Web objects can effectively mask EWM measurement signatures and hinder manipulation. Any attempt to systematically delay responses is detectable and necessarily degrades a server's performance.

[3]The icon (favicon.ico) will be requested by the browser every time the browser opens a page from the Web site.

7 Related Work

We presented EWM's related methodologies in detail in Section 2 above. We outline other related work below.

Previous work [24, 37, 45] has discussed benchmarking Web servers in terms of behavior, performance, and capacity. However, contrary to EWM, these research efforts also assume that the Web servers' log is available for analysis. In [27, 40], the authors proposed methodologies to generate Web traffic and evaluate Web servers' capacity. Our work here is different because we address the problem of measuring the "live" traffic load of a Web server. Baryshnikov *et al.* [28] present a predictor for forecasting Web servers' congestion scenarios, yet this work is limited to predicting hotspots where Web traffic is congested. EWM can measure the Web traffic in real time independently from a Web server's congestion status, *i.e.*, whether it is congested or not. Urgaokar *et al.* [43, 44] analyze the performance of a multi-tier Web service under various workloads and they focus on evaluating the response time of Web requests. EWM is able to decouple response time into service time and network latency. In addition, EWM estimates the number of active connections to a Web server by monitoring its service time.

8 Conclusions

In this paper, we presented the design and implementation of EWM, a novel measurement methodology for monitoring the popularity of Web services. Contrary to existing approaches, EWM neither requires network or Web server administrative privileges nor large-scale Web clients' support, which places it at a unique position in the Web traffic monitoring design space. By using a combination of TCP ping and HTTP `HEAD` probes, EWM is able to accurately measure Web servers' service time by effectively removing the network latency component. Furthermore, using artificial connections and an MIMD probe control, EWM can quickly converge to the knee point of the Web servers' accept queue and measure the active connections to the Web servers. Our evaluations show that EWM can attain a valid measurement on an average of 13.5 seconds with a negligible measurement overhead of 4.37 kb/s. These properties enable EWM both to monitor popularity trends over short time scales and to achieve extraordinary scalability properties.

We evaluated EWM in a number of realistic scenarios, including a three-tier Web service, a real-world blogging scenario, and an online auction scenario. In all cases, we demonstrated EWM's ability to achieve a desirable point between accuracy and fast convergence. Finally, we conducted larger-scale case studies and demonstrated EWM's practicality in monitoring antivirus and

open source software vendors, by measuring the corresponding server clusters. These measurements provide the proof-of-concept results and represent only the first two among many other Web services that we plan to evaluate. Most importantly, our key contribution lies in devising a deployable system capable of directly, independently, accurately, and fairly assessing Web servers' popularity. Such a system has the potential to earn high credibility and public trust, hence benefit the Internet community as a whole.

References

[1] Akamai. http://www.akamai.com/.

[2] Alexa.com. http://www.alexa.com/.

[3] avast! antivirus. http://www.avast.com/index.

[4] Avira antivirus. http://www.avira.com/en/pages/index.php.

[5] AWSats. http://awstats.sourceforge.net/.

[6] Can you trust Alexa and Google Trends? http://www.markpack.org.uk/can-you-trust-alexa-and-google-trends/.

[7] Compuware Vantage. http://www.compuware.com/solutions/vantage.asp.

[8] comScore.com. http://www.comscore.com/.

[9] Definition of web bug. http://www.pcmag.com/encyclopedia_term/0,2542,t=Web+bug&i=54280,00.asp.

[10] Google analytics. http://www.google.com/analytics/.

[11] Google trends. http://trends.google.com/websites.

[12] Haproxy: The reliable, high performance tcp/http load balancer. http://haproxy.1wt.eu/.

[13] How does load balancing work? http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094820.shtml.

[14] Measuring web traffic. http://www.ibm.com/developerworks/web/library/wa-mwt1/.

[15] Nielsen, comScore Stat Accuracy Questioned. http://www.betanews.com/article/Nielsen-comScore-Stat-Accuracy- Questioned/1177365481.

[16] Omniture.com. http://www.omniture.com/.

[17] Rubis: Rice university bidding system. http://rubis.ow2.org/.

[18] Tcp tuning. http://people.redhat.com/alikins/system_tuning.html#tcp.

[19] tcpdump. http://www.tcpdump.org.

[20] Tealeaf CX. http://www.tealeaf.com/products/cx.php.

[21] Web-stat.com. http://www.web-stat.net/.

[22] The webalizer. http://www.webalizer.org/.

[23] Wireshark. http://www.wireshark.org.

[24] AMZA, C., CH, A., COX, A. L., ELNIKETY, S., GIL, R., RAJAMANI, K., CECCHET, E., AND MARGUERITE, J. Specification and implementation of dynamic web site benchmarks. In *IEEE WWC* (2002).

[25] ANDERSON, T. E., COLLINS, A., KRISHNAMURTHY, A., AND ZAHORJAN, J. PCP: Efficient endpoint congestion control. In *USENIX NSDI* (2006).

[26] AYANI, R. Comparison of load balancing strategies on cluster-based web servers. *Simulation 77*, 5-6 (2001), 185–195.

[27] BANGA, G., AND DRUSCHEL, P. Measuring the capacity of a web server under realistic loads. *World Wide Web 2*, 1-2 (1999), 69–83.

[28] BARYSHNIKOV, Y., COFFMAN, E., PIERRE, G., RUBENSTEIN, D., SQUILLANTE, M., AND YIMWADSANA, T. Predictability of web-server traffic congestion. In *IEEE WCW* (2005).

[29] BENT, L., KOLETSOU, M., AND VOELKER, G. The Medusa proxy: Parallel connections under two browsers. http://www.sysnet.ucsd.edu/medusa/.

[30] BOURKE, T. *Server load balancing*. O'Reilly Media, 2001.

[31] BRESLAU, L., KNIGHTLY, E. W., SHENKER, S., STOICA, I., AND ZHANG, H. Endpoint admission control: Architectural issues and performance. In *ACM SIGCOMM* (2000).

[32] CHOI, H.-K., AND LIMB, J. O. A behavioral model of web traffic. In *IEEE ICNP* (1999).

[33] FLOYD, S., AND JACOBSON, V. On traffic phase effects in packet-switched gateways. *Internetworking: Research and Experience 3*, 3 (Sept. 1992), 115–156.

[34] GUMMADI, K., MADHYASTHA, H., GRIBBLE, S., LEVY, H., AND WETHERALL, D. Improving the reliability of Internet paths with one-hop source routing. In *USENIX OSDI* (2004).

[35] JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites. In *WWW* (2002).

[36] KANDULA, S., KATABI, D., JACOB, M., AND BERGER, A. Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In *USENIX NSDI* (2005).

[37] LI, Z., ZHANG, M., ZHU, Z., CHEN, Y., GREENBERG, A. G., AND WANG, Y.-M. Webprophet: Automating performance prediction for web services. In *USENIX NSDI* (2010).

[38] LIU, Z., NICLAUSSE, N., AND JALPA-VILLANUEVA, C. Traffic model and performance evaluation of web servers. *Performance Evaluation 46*, 2-3 (Oct. 2001), 77–100.

[39] MACIA, G., WANG, Y., RODRIGUEZ, R., AND KUZMANOVIC, A. ISP-enabled behavioral ad targeting without deep packet inspection. In *IEEE INFOCOM* (2010).

[40] MOSBERGER, D., AND JIN, T. httperf - a tool for measuring web server performance. *SIGMETRICS Performance Evaluation Review 26*, 3 (1998), 31–37.

[41] ROTH, A., AND OCKENFELS, A. Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the Internet. *American Economic Review 92*, 4 (2002), 1093–1103.

[42] STEIGLITZ, K. *Snipers, shills, & sharks: eBay and human behavior*. Princeton University Press, 2007.

[43] URGAONKAR, B., PACIFICI, G., SHENOY, P., SPREITZER, M., AND TANTAWI, A. An analytical model for multi-tier internet services and its applications. In *ACM SIGMETRICS* (2005).

[44] URGAONKAR, B., PACIFICI, G., SHENOY, P. J., SPREITZER, M., AND TANTAWI, A. N. Analytic modeling of multitier internet applications. *TWEB 1*, 1 (2007).

[45] VEAL, B., AND FOONG, A. Performance scalability of a multicore web server. In *ACM ANCS* (2007).