Advance Access publication on 8 November 2019
doi: 10.1093/comjnl/bxz081

Power Saving Proxies for Web Servers

Karl J. O'Dwyer[1], Eoin Creedon[2], Mark Purcell[2] and David Malone[1,*]

[1]*Hamilton Institute, Maynooth University, Kildare, Ireland*

[2]*IBM Research Ireland*

*****Corresponding author: David.Malone@mu.ie*

Electricity is a major cost in running a data centre, and servers are responsible for a significant percentage of the power consumption. Given the widespread use of HTTP, both as a service and a component of other services, it is worthwhile reducing the power consumption of web servers. In this paper we consider how reverse proxies, commonly used to improve the performance of web servers, might be used to improve energy efficiency. We suggest that when demand on a server is low, it may be possible to switch off servers. In their absence, an embedded system with a small energy footprint could act as a reverse proxy serving commonly-requested content. When new content is required, the reverse proxy can power on the servers to meet this new load. Our results indicate that even with a modest server, we can get a 25% power saving while maintaining acceptable performance.

Keywords: power aware computing; power saving; HTTP; Web server; reverse proxies.;

Received 11 September 2018; Revised 6 April 2019; Editorial Decision
Handling editor: Gerard Parr

1. INTRODUCTION

IT now contributes measurably to the global consumption of electricity. While this is less than other sectors [1] and progress has been made, there is room for improvement in areas such as on-demand use of resources, trading energy usage and quality of service [2]. New server technologies promise to lower power consumption when resources are idle. Servers are specified to meet or exceed current peak demands; however, common power-saving techniques are unable to power the server off while idle, as the server must remain responsive to new requests. Empirical tests show that while powered off or sleeping, servers consume significantly less power than in the lowest-power idle states (e.g. [3, 4]).

Mechanisms for temporarily turning servers on and off do exist. Recent server hardware supports the Advanced Configuration and Power Interface (ACPI) power-saving modes, which were previously only available on laptop and desktop computers. These include methods to suspend to disk (i.e. hibernation) and suspend to RAM, which are low power modes with quick recovery to normal operation. The Wake-on-LAN (WoL) standard for remote activation of devices offers a convenient method to initiate recovery remotely.

Thus, if we can overcome the adverse effect on availability resulting from putting a server to sleep, then it may be practical to save energy. The use of reverse proxies for website acceleration or load balancing is relatively commonplace. In this paper we consider using a low-power device acting as a reverse proxy. It will have the additional ability to shut down the server when demand is low, serve cached content while the server is sleeping and wake the server if new content is required. In contrast to previous papers, which have considered how to reduce power usage when the service is provided in the cloud or on a large-scale in a data-centre (see Section 2 for a review), we are targeting services typically provided by a single server which may have low-demand periods (e.g. in-house servers at night, low-demand on-site hosted web servers, . . .).

We explore this approach by developing a small testbed that allows us to replay web access patterns, estimate energy savings, etc. We review some related work in Section 2. In Section 3, we describe the web access patterns that we observe on a campus web server and their implications for designing a power-saving scheme. In Section 4, we describe the relevant power-management features and measure their power usage. We then discuss the limits of possible power savings without introducing additional delays in Section 5. This leads us to the design of a power-saving scheme in Section 6 and our evaluation of this scheme in Section 7. In Section 8 we discuss our results and conclude in Section 9. This paper is an extended version of the conference paper [5], which presents preliminary results on web access patterns and the limits of power saving schemes.

2. RELATED WORK

Though energy saving has long been of interest for clients such as laptops (e.g. [6]), desktops (e.g. [7, 8] and phones (e.g. [9]), there has been a thread of interest in applying techniques to servers, including an early call to apply them to web servers [10]. Various specific facilities have been used to improve energy efficiency, often controlled by the host operating system. For example, Dynamic Voltage and Frequency Scaling (DVFS) can reduce the energy consumption of the CPU and other components and is often controlled by the operating system (e.g. [11]). *Energy proportionality* is the idea that the amount of energy used should be proportional to the amount of work done [12], but this can be challenging to achieve with some computer components, such as disks, that are either on or off. Sleep states have long been used for desktop computers and mobile devices, but they have proven useful on servers too. For example, Gandhi *et al.* [3] make the case for sleep states on servers, evaluating their use in a cluster of 24 servers. Of course, other features can have an impact on energy consumption, from the programming language used to implement the system [13], the protocols in use at the client [14] or the server [15], to the use of modern devices such as GPUs (Graphic Processing Units) [16].

Energy usage in data centres and the cloud has generated a large literature. The ability to consolidate workloads [17] has provided the facility to reduce the number of idle services. A common thread in many energy-reduction schemes is to model the power usage of the data centre or cloud and combine this with a characterization of the demand (e.g. [18, 19]). More recent works take such schemes and build resource management frameworks for data centres [20]. In a similar, vein Xu and Li [21] propose an energy efficient framework for resource allocation in the cloud, and others have proposed schemes aiming for energy proportionality for large-scale systems consisting of thousands of servers [22]. Simulation environments for understanding energy usage of cloud services have also been built and tested [23–27]. Surveys of research on energy efficiency in the cloud [28] and data centres [29] have been conducted, including surveys of specific applications, such as computational work [30], taxonomies of the problems being addressed [31], broader strategies for energy and carbon efficiency of data centres [32] and even surveys of the surveys [33].

Another more application-specific way to consider this problem is to consider the content delivery networks (CDNs) that are used by large-scale providers of web content. These works consider factors impacting energy consumption (e.g. [34]) and how CDNs can be managed for energy efficiency (e.g. [35]). This can include powering off devices in a CDN network (e.g. [36, 37]) and the trade off between energy and QoS (e.g. [38]).

In this paper, we share common elements with previous work, such as exploiting typical traffic patterns and measuring/modeling energy usage in order to obtain savings. Our focus, however, is away from the data centre and instead on services that might be provided by individual servers, rather than large-scale systems, where low-demand periods can be still exploited to save energy. Particularly, by reducing baseline energy usage when the system is close to idle we aim to move the system closer to energy proportionality.

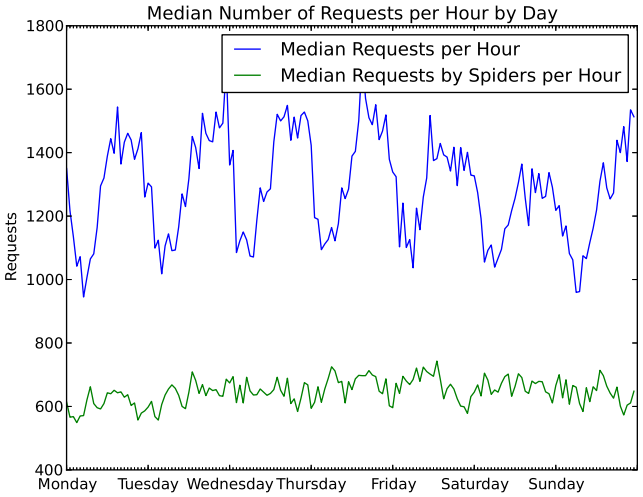

FIGURE 1. Median number requests per hour by day, sample web logs.

3. INVESTIGATION OF TRAFFIC

Our aim is to exploit patterns in web traffic in order to turn off web servers when they are not required. There has been considerable work to characterize web traffic (e.g. [39–44]), and it is known that web access patterns are bursty.

A reverse proxy is a web server that accepts requests from clients and forwards them to a *back-end* server, which stores or generates all the website's available content. The reverse proxy caches the content as it is served, and uses the cached content to answer requests when possible. As websites often have 'hot' static content [45], or content that is expensive to generate but can be cached once generated, reverse proxies can often result in performance improvements. We use Varnish [46] as a reverse proxy. The reverse proxy's ability to cache content, and so to save power, will depend on the details of the accesses. Consequently, we will design and assess our scheme using actual requests from a campus web server.

This web server hosts the websites of around 400 student clubs, societies and individuals. It has a variety of content and is accessed frequently by those on and off the campus, and exhibits a mix of web content and access patterns. In total, it amounts to over 77GB of content in 400 000 files, excluding content stored in databases. Frequently-accessed content represents a considerably smaller subset of this total. The server is not busy, typically serving around 30 000 requests per day, but the volume of content should make caching more challenging.

We looked at the median and mean number of requests per hour over 270 days worth of data and grouped them by a

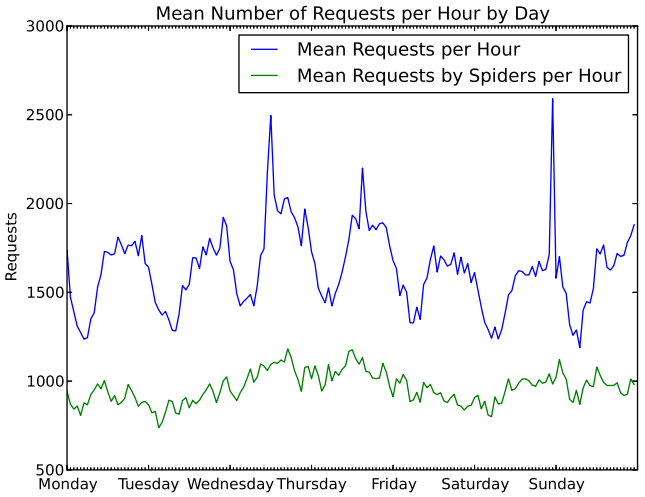

FIGURE 2. Mean number requests per hour by day, sample web logs.

TABLE 1. Strings used to identify common spiders.

Googlebot	Slurp	Baiduspider
bingbot	urlresolver	Speedy Spider
Sosospider	Sogou web spider	Gigabot

specific hour during a week. The median is shown in Fig. 1, with the noisier mean values in Fig. 2. We do not see periodic activity, but a diurnal pattern emerges, showing significant variation throughout the day.

Further inspection of the data reveals that a significant number of requests are from web spiders [47].[1] Based on a manual inspection of the User Agent field of the log file, and consulting lists of common spiders, we found that matching the list in Table 1 allowed us to identify the majority of requests made by spiders to this site. We randomly sampled requests that did not match this list, and inspected them manually. Only $\sim$ 5% of the remaining requests came from suspected spiders.

We calculate the median and mean request rate from spiders in this list and also show this median and mean in Fig. 1 and Fig. 2, respectively. We see that a significant number of requests are actually from these spiders and that this traffic does not exhibit the same diurnal pattern. This suggests that it may be useful to handle web traffic from spiders as a special case.

If we want a server to sleep between requests, then an important factor is the period between requests, which we estimate using the length of gaps between logged requests (logged at a resolution of 1 second in Combined Logfile Format). The length of these gaps will indicate whether it may be possible to switch off the web server between requests, or if such opportunities are limited.

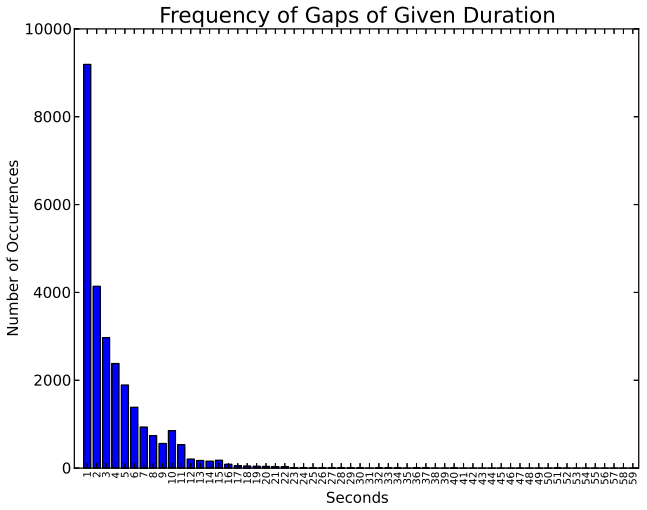

FIGURE 3. Number of gaps of a particular duration between requests. We omit gaps of duration <1 second.

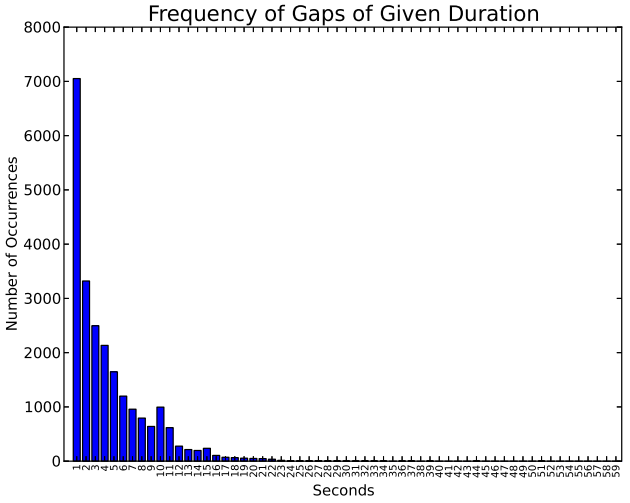

FIGURE 4. Number of gaps of a particular duration between requests to back end, omitting requests served by reverse proxy. We omit gaps of duration <1 second.

Using a subset of our data, amounting to 40305 requests over 28 hours, we look at the distribution of the gaps. Figure 3 shows the frequency of the gap of a particular duration. As we expect, gaps are typically quite short, which limits our chances to turn a server off and on without impacting on web traffic.

To consider the impact of caching on the gaps between requests to the (back-end) web server we replayed the requests to the campus server using Varnish [46] as a reverse proxy to cache the content.[2] The resulting distribution of gap lengths is shown in Fig. 4. We see an increase in the number of long

[1] A web spider, sometimes called a robot, a bot or a crawler, is an automated system that loads web pages [48, 49], e.g. search-engines' crawlers.

[2] The cache starts empty. We use the default Varnish configuration.

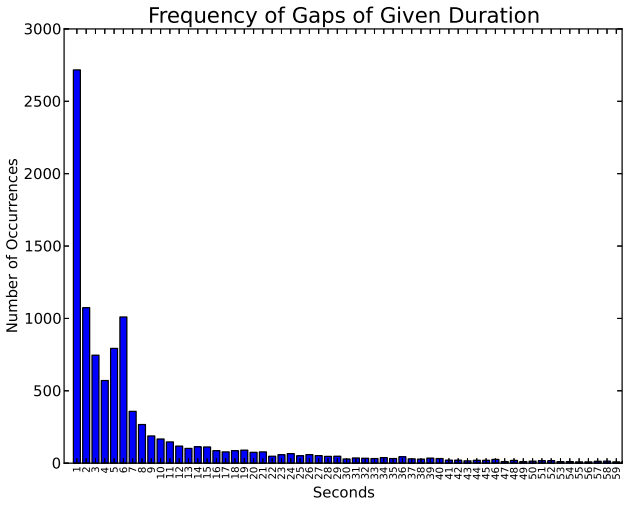

FIGURE 5. Number of gaps of a particular duration between non-spider requests. We omit gaps of duration <1 second.

duration gaps, representing an increase in opportunities to put the server to sleep.

For comparison, we also consider the distribution of gaps between requests when we omit requests from spiders. We do this on the basis that a spider does not usually need the content immediately, and indeed, some spiders can be told to make their requests later using a HTTP 503 response with a Retry-After header [50–52]. The results are shown in Fig. 5. We now see that the tail of the distribution of gap durations has thickened, which suggests a better chance of powering the server off without impacting on user requests. Comparing these results with Fig. 4, it appears that, for this log file, smart handling of spiders might have a bigger impact than caching of commonly-accessed content.

4. POWER STATES AND RECOVERY

Various systems are available for controlling the power state of servers. In particular, we will make use of an interface for putting the system into a low power state (ACPI) and then waking it at some later point (WoL).

ACPI is a standard that aims to consolidate all power management and configuration standards [53]. The ACPI standard defines a number of Power States, from G0 (active) to G3 (mechanical off). The sleeping state, G1, is subdivided into 4 states (S1–S4). For us, S3 and S4 are interesting as they define Suspend to RAM and Suspend to Disk respectively. As we will see, in these states servers consume almost as little power as when powered off. With operating system support, the power state of a server can be changed.

WoL is an industry standard introduced by IBM and Intel for remotely powering on computers. It requires a compatible network interface card which remains powered on after the computer is powered off. A number of different signals can be used to wake a computer, based on hardware support, including PHY activity, ARP and broadcast, unicast and multicast messages. The most commonly supported signal used to wake up a computer is called the *Magic Packet*, a broadcast frame with a payload that is 6 bytes set to 255 followed by the target computer's MAC address repeated 16 times. WoL can usually be configured via the BIOS/firmware or operating system.

To illustrate these power management features, we investigate the power consumption of the device we use as a server. We investigated a number of methods of measuring power usage, such as the Maplin^{TM} 200MU-UK (which gives a manual reading of total energy usage) or a APC^{TM} AP7900 PDU (which gave regular power usage measurements by SNMP, but gave inaccurate readings at low power usage). We decided to make our measurements using a Current Cost EnviR and a Current Cost Individual Appliance Monitor. The energy consumption for each device is supplied by the monitor and logged via USB using a Python script every 7 seconds. We describe how we model the energy use of our servers below. For these systems, we also measure the time needed to put the system to sleep and the time to wake after a WoL message. This will influence the responsiveness of our scheme. We call the sum of these the *turn-around-time*. We also characterize the turn-around-time below.

We characterize our hardware in Table 2. For comparison, we also show the power consumption of a Soekris net5501. We will use this as a reverse web proxy during quiet periods.

> MSI Server: The server used was a custom-built server based on an MSI 'Military class' motherboard, an Intel i7 2700k processor and 16GB of RAM. We used the on-board 1G Ethernet for a network connection. We tested several other servers of this and older generations and found that this system had good support for both WoL and use of ACPI power states. The power supply was rated at several hundred Watts, though in practice it used considerably less when operating as a web server. We ran Ubuntu 12.04 LTS on the server.
>
> Soekris net5501: The Soekris net5501 is a single board PC based around the AMD Geode^{TM}, using a max of 20W, but typically much lower, as we see in Table 2. We ran an older version of Ubuntu, 8.04.4 LTS, with smaller resource requirements on this device. We did not use or report on the sleep/wake features of the net5501, as we intended to use this device as the reverse proxy.

In reviewing older server hardware, we found that WoL and ACPI support was less consistent. For example, after upgrading the firmware on a Dell$^{®}$ PowerEdge^{TM} 1800 we found

TABLE 2. Measured power profile for devices.

	On power usage (W)	Off power usage (W)	Switch cost (J)	Sleep time (s)	Time to recover (s)
MSI™ server	43.3	1.6	170.7	3.7–4.3	0.1–0.2
Soekris net 5501	5	NA	NA	NA	NA

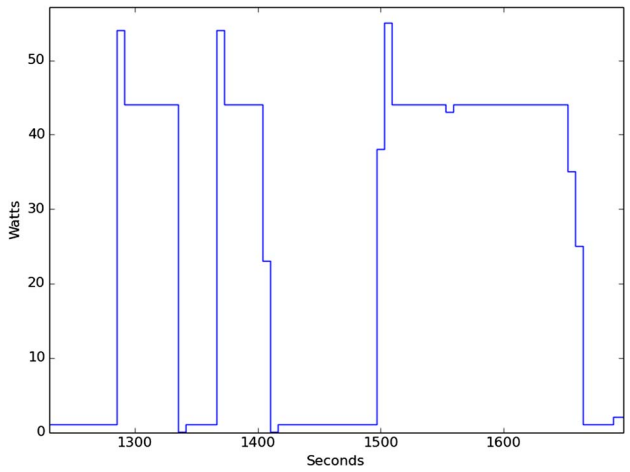

FIGURE 6. Power consumption for a server when sleeping, waking and powered on.

WoL and ACPI were available. However, as only suspend to disk was possible, wake times were over 1 minute. The Dell® Optiplex™ 755 desktop system supported suspend to RAM, with much better wake times, however we found that WoL did not always wake the system, and though capable of running a web server, the system was not typical of server-class hardware.

4.1 Modelling power and energy usage

Figure 6 shows the power usage for our server over a number of sleep/wake/powered-on cycles. We can see that there is a pattern, which suggests roughly constant power usage when sleeping or powered on/off. There is a brief spike in power usage when the machine wakes. Based on this, we model the server as having a power usage P_{on} while on, a power usage P_{off} when off and there is a cost for each switch-on event of E_{switch}. From Fig. 6 we see that these quantities are not fixed, but in fact random.

We model these by measuring the power used by the test bed while it performs a series of sleeps and wakes of variable length in an experiment labelled i. For each experiment we record four variables: the total amount of time awake T_{on_i}, the total amount of time sleeping as T_{off_i}, the number of times it switches on or off N_{switch_i} and the total energy used E_{total_i}. Our model suggests that in expectation we have

$$T_{\text{on}_i}\mathbb{E}[P_{\text{on}}] + T_{\text{off}_i}\mathbb{E}[P_{\text{off}}] + N_{\text{switch}_i}\mathbb{E}[E_{\text{switch}}] = \mathbb{E}[E_{\text{total}_i}] \quad (1)$$

We then want to estimate the power used while the server is on and off, and the energy overhead for a switch between these two states. Over a number of runs we can write this as

$$\begin{pmatrix} T_{\text{on}_i} & T_{\text{off}_i} & N_{\text{switch}_i} \\ \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \mathbb{E}[P_{\text{on}}] \\ \mathbb{E}[P_{\text{off}}] \\ \mathbb{E}[E_{\text{switch}}] \end{pmatrix} \approx \begin{pmatrix} E_{total_i} \\ \vdots \end{pmatrix}. \quad (2)$$

We can then use the least squares estimator for $\mathbb{E}[P_{\text{on}}]$, $\mathbb{E}[P_{\text{off}}]$ and $\mathbb{E}[E_{\text{switch}}]$ given E_{total_i},

$$\begin{pmatrix} \mathbb{E}[P_{\text{on}}] \\ \mathbb{E}[P_{\text{off}}] \\ \mathbb{E}[E_{\text{switch}}] \end{pmatrix} \approx (A^T A)^{-1} A^T \begin{pmatrix} E_{total_i} \\ \vdots \end{pmatrix}, \quad (3)$$

where A is the matrix of on/off times and switch counts. The results of performing these estimates for our server are shown in Table 2.

4.2 Server wake/sleep times

The length of time required to put a server to sleep and to wake it again is also important to our scheme. These determine the minimum amount of time we can sleep for, and also how long it will take to answer an uncached request that arrives while the server is asleep. We describe how we measure both times below using ICMP ping packets to determine if the IP stack of the server is operational.

To measure the time that it takes the server to be put asleep, we issue a sleep command to the server via ssh. At the same time, we ping the server once every 100 ms until we receive no response. The number of pings between issuing the ssh command and the last received ping gives us the time to sleep in tenths of a second. Note, that there is some timing overhead associated with the ssh connection, which would not be present if a more lightweight way of issuing the sleep command was used. To estimate this, we also timed how long it took ssh to run a null command.

To estimate the wake-up time, we also send a ping to the server ever 100ms and then send a WoL magic packet. We then observe the amount of time between issuing the magic packet and when the server starts to respond. In order to avoid issues

with ARP entries timing out, we manually set the ARP entries for the server. Again, this gives us an estimate of the time to wake the system to the nearest tenth of a second.

The results of estimating both the wake and sleep times for our server are shown in Table 2.

5. IDEALIZED POWER SAVING MODEL

In this section, we show how to estimate the possible power savings for a web server, given information about power usage, gaps between requests and how quickly it can be turned on and off. Consider an idealized situation, where the server could somehow determine when the next request will arrive. After serving each request, it could see if the time to the next request is greater than its turn-around-time. If so, the server goes immediately to sleep and schedules a wake up just in time to serve the next request. This would allow the server to serve *all* requests without introducing *any* delays waking the server, while sleeping for the maximum time possible. However, as we would only like to consider sleeping if we get a resulting power saving, and as we will see in Section 6.3, this can also be converted into a minimum threshold on the sleep time.

We can calculate the power saving given by this idealized scheme. Let $(t_i)_{i=1..N}$ be the sequence of gaps between requests. We can find T_o and T_s, the time that the server spends on or sleeping respectively. For this scheme,

$$T_o = \sum_{i=1, t_i < t_{\text{thr}}}^{N} t_i$$

and

$$T_s = \sum_{i=1, t_i \geq t_{\text{thr}}}^{N} t_i,$$

where t_{thr} is the threshold given by the minimum turn-around-time and the requirement for power saving. Total energy usage, in kWh, will then be

$$\frac{T_o P_{\text{on}} + T_s P_{\text{off}} + N E_{\text{switch}}}{3.6 \times 10^6},$$

where N is the number of switches, given by the number times $t_i < t_{\text{thr}}$.

Using this model, we can assess the power saving possible without introducing extra delays. Figure 7 shows the total energy consumption possible as a function of the threshold time, t_{thr}, assessed for the original sequence of requests served by the web server. We see that for the original log file, for a turn-around-time of 20s or more, there are few opportunities to sleep, and the total power usage is similar to having the server always on. However, turn-around-times of 2–5 s actually result in significant savings.

Following our observations from Section 3, we consider the impact of spiders and caching. First, Fig. 7 also shows

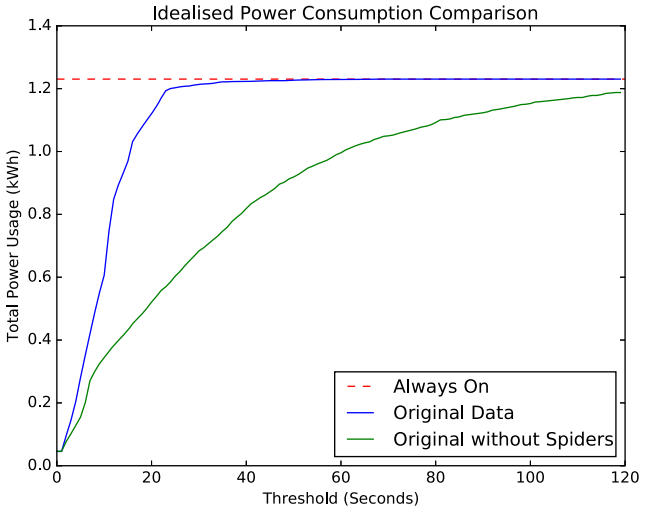

FIGURE 7. Power saving for the idealized scheme that results in no delay, as a function of the turn-around time.

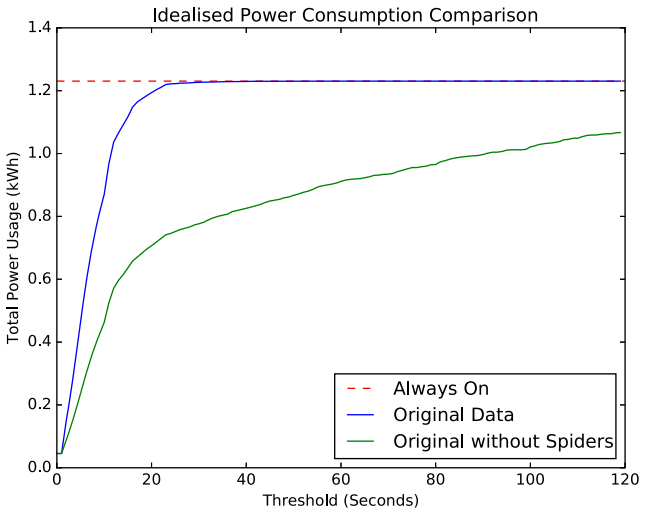

FIGURE 8. Power saving for idealized scheme, answering requests that cannot be responded to by reverse proxy as a function of the turn-around time.

the results if we are willing to ignore requests from spiders. Here the situation is much more promising. With our idealized scheme, a turn-around-time of 20 s allows a reduction of energy consumption to around one third of the consumption for a server that is always on.

Second, we consider the impact of caching in Fig. 8. Here, we use the idealized power-saving model on requests that are passed by a Varnish cache to the the back end server. This eliminates the need for the backend to serve the content cached by the reverse proxy. As expected, based on our discussion in Section 3, we see a saving over answering all requests; however, the improvement is not as large as the saving for ignoring requests from spiders. We also see that combining

caching with the special handling of requests from spiders results in useful gains.

6. DESIGN OF A PRACTICAL POWER SAVING STRATEGY

In this section we will consider how to design a practical scheme to allow web servers to sleep. We consider when to power the server down, when to power it up and, as the scheme does not have advance knowledge of the requests, how to handle requests that arrive when the server is powered down.

6.1 Architecture

The system we consider is a high-power web server with a low-power reverse proxy (though other configurations are discussed in Section 8). The reverse proxy manages the power state of the web server via ACPI and WoL. We assume that the low-power proxy does not cause performance problems during periods of high load. In practice, this issue can be overcome by switching to directly serving requests (via DNS or ARP/IP mapping) or switching in a higher-power proxy at higher loads. Similarly, we assume that caching is managed in the usual way by Varnish, with parameters set as described in Section 7.1.

6.2 Power-on decisions

Power-on decisions must be made by the reverse proxy, as the server is not available to make decisions itself. Thus, we describe the power-on decisions in terms of the operation of the reverse proxy. Of course, if the web server is on, we configure the reverse proxy to operate as usual, responding to queries from the cache or making requests to the back end.

If a server is off, we trigger power-on requests based on the arrival of server requests. Naturally, if the request can be served from the cache, then there is no need to wake the back end. Based on the discussion in Section 3, we believe there may be an advantage to handling requests from spiders in a special way. If the back end is off and the request is from a spider, we also choose not to wake the back end and instead issue a 503 response, indicating that the spider should return later. We suggest that this time could be chosen to be a known-busy hour of the day or a period specially chosen to facilitate website crawling by spiders. The logic is summarized in Algorithm 1.

Algorithm 1 Decision process to wake sleeping back ends.

```
if Request in Cache then
    Serve request from Cache
else if Back end Server Available then
    Serve request from back-end server
else if Request from Spider then
    Issue 503 to request spider
else
    Wake back end
end if
```

6.2.1 *Implementation*

In our implementation, the server's state is tracked by Varnish. Algorithm 1 can easily be implemented using Varnish's VCL configuration language to tag requests from spiders in `vcl_recv`, and deciding if the server should be woken in `vcl_miss`.

6.3 Power-off decisions

In practice, the power-off decision could be made by either the back-end web server or the reverse proxy. We considered a number of possible strategies for making this decision, including forecasting demand based on previous weekly/diurnal patterns, monitoring the hit rate in the reverse proxy's cache, or using a learning algorithm. We will consider a scheme that aims to achieve power savings on average while also achieving a target delay for serving requests.

One possible improvement that applies to most schemes is to ensure server sleep time isn't interrupted by requests from web crawling agents used to index websites for search engines. This can be achieved by responding to all requests from such agents with either cached-content, no update or a HTTP 503 error (this should mitigate any negative effect on search ranking). Consideration should also be given to client-side caching for supported clients as a means to reduce the number of possible requests to the server.

6.3.1 *Expectation of power saving*

An important consideration for switching off is, *will turning off save power?* Ideally we would like the power used to turn off and then power back on again to be less than the power used for turning on, i.e. $TP_{\text{off}} + E_{\text{switch}} < TP_{\text{on}}$, where T is time between requests, and P_{off}, P_{on} and E_{switch} are the energy usage parameters described above. Since these quantities, particularly T, are random quantities, we can instead ask to save power on average, so we take the expectation,

$$\mathbb{E}[TP_{\text{off}} + E_{\text{switch}}] < \mathbb{E}[TP_{\text{on}}] \tag{4}$$

$$\mathbb{E}[T]\mathbb{E}[P_{\text{off}}] + \mathbb{E}[E_{\text{switch}}] < \mathbb{E}[T]\mathbb{E}[P_{\text{on}}] \tag{5}$$

This can be further simplified to give us this lower bound, below which we should not send the server to sleep:

$$\mathbb{E}[T] > \frac{\mathbb{E}[E_{\text{switch}}]}{\mathbb{E}[P_{\text{on}}] - \mathbb{E}[P_{\text{off}}]} \tag{6}$$

provided that $\mathbb{E}[P_{\text{on}}] - \mathbb{E}[P_{\text{off}}]$ is positive, which we expect for realistic on/off power values.

To implement this condition, we estimate the power usage parameters, as described in Section 4.1, and use a window of 10 requests to estimate the current spacing to the next request T.

6.3.2 *Managing delay*

If we do not mind delaying requests, then putting the server to sleep for extended periods can save a lot of power. However, we suppose that we have a SLA that requires that no more than a fraction f_{SLA} of the requests can be delayed by more than a delay δ over some window.

Suppose that at some time the number of requests that will arrive in a turn-around period is $\mathcal{K}$. If $\mathcal{D}$ is the number of requests that have exceeded the delay threshold δ and $\mathcal{T}$ is the total number of requests seen in the current window, then, to achieve the SLA, we require

$$\mathcal{D} + (1 + \mathbb{E}[\mathcal{K}]) < f_{\text{SLA}}(\mathcal{T} + (1 + \mathbb{E}[\mathcal{K}])). \tag{7}$$

In this case, we know $\mathcal{D}$ and $\mathcal{T}$, however $\mathcal{K}$ is random and unknown. So, by taking expectations, we get a condition on $\mathbb{E}[\mathcal{K}]$:

$$\mathbb{E}[\mathcal{K}] < \left(\frac{f_{\text{SLA}}\mathcal{T} + f_{\text{SLA}} - \mathcal{D} - 1}{1 - f_{\text{SLA}}}\right) \tag{8}$$

provided that $f_{\text{SLA}} \leq 1$

The implementation of this condition is relatively simple, and requires the server to maintain a counter of the total number of requests $\mathcal{T}$ and a counter of delayed requests $\mathcal{D}$. Estimating the expected number of requests in the turn around time $\mathcal{K}$ is relatively simple, by maintaining the request rate over a window. Note that we expect that maintaining this condition will increase power usage, which is in line with other studies considering power usage and SLAs [2, 54].

6.3.3 *Power cycling cost and debounce*

Our scheme, in practice, limits the amount of switching between on and off states. However, a server might want to limit the amount hardware stress due to power cycling or bouncing between on and off states because of conflicting on and off conditions. We implement two simple conditions: first, we will not power off the server for 60 seconds after a power-on, to avoid excessive power cycling; second, we will wait at least 5 seconds after the most recent request before powering the server down, to allow slow clients to request all the content for a page load.

6.3.4 *Implementation*

For our evaluation, we prototype a small daemon in Python that runs on the reverse proxy. Varnish uses signals to notify the daemon of incoming requests and the daemon reads Varnish's log files to count requests meeting the delay bound. It also makes the server's current state available to Varnish via shared memory to ease the implementation of Algorithm 1.

6.4 Mitigating mistakes

We note that a power-saving scheme does not know when requests will arrive so it will ultimately make some mistakes. If requests from users are delayed while waiting for a server to awaken, this could have a negative impact on their experience of using the service, or might result in a HTTP connection timeout. In the case that the reverse proxy detects that is must wake the server, we propose a method to combat this. If the content is delayed, alternative web pages could be presented by the proxy, such as a splash screen or a loading bar. Alternatively, an advertisement or suggestion that the user take a short survey might be offered.

Note that we must only stall the user for ∼5 seconds in order to wake a server from the suspend-to-RAM state, when the user can be redirected to their originally requested content. We have implemented this last method as an addition to Varnish. If a request requires a server to power on, the end user is served a page asking them to take part in a survey and giving them a countdown until they will be redirected to their requested content. We call this method 'Survey'.

6.5 Other considerations

Reverse proxies are generally used to increase throughput or performance. To decrease power usage, most of our tests are conducted with a low-power device, which may actually decrease performance. We evaluate the impact of this in Section 7 by studying actual request delays. However, we also assess the possibility of using a high-performance embedded device in Section 7.4 as an alternative.

7. RESULTS

7.1 Test system

Our prototype test system consists of three components: a client, a reverse proxy and a server. The client makes requests to the reverse proxy or server using a group of Python scripts that replay a given Apache access log in real time (i.e. requests are spaced as in the log file). These scripts record the HTTP status of the request responses and the delay in serving the content.

For a reverse proxy, we use a Soekris net5501 embedded computer, with power usage detailed in Table 2. The system runs a version of Varnish Cache (Version 3.0.1), which has been modified to allow the operation described in Section 6.1. Varnish has been configured with a 128 MB cache space. The connection timeout to the back end is set to 10s. The server runs Apache 2.2.2 and is a MSI$^{\text{TM}}$ -based system whose power requirements are detailed in Table 2.

To implement the power-saving scheme from Section 6.1, we send UDP packets between the reverse proxy and the server. We found that all the WoL implementations we tested sometimes entered a state where wake requests were ignored. To work around this, we initially tried using an optocoupler attached to the Soekris net5501 GPIO pins to 'press' the power button on the server. While this mechanism was more reliable than using a WoL packet, we still found the servers occasionally in a state where they would not wake. Our conclusion is that

either the Linux hibernation code or the ACPI implementation on the servers we tested is not dependable enough to run our scheme.

Consequently, for our experiments to assess the performance of the scheme, instead of putting the server to sleep, we set up a firewall rule that blocking HTTP requests. These rules are removed, after a delay to simulate the wake time, when the system is woken. We use 5 seconds as the delay, to overestimate the full wake/sleep cycle from Table 2. We then estimate the power usage using the log of sleep/wake events and the power model from Section 4.1. One advantage of this method is that we can vary the turn-around-time and power model to estimate performance of other systems.

7.2 Evaluation

To evaluate the performance and power usage of our scheme, we replay the 28-hour section of log file described in Section 3. We consider five different configurations of the test system, chosen to offer some insight into the performance trade-offs involved in powering the system down. In each case the cache starts empty.

No Varnish: The server is always powered on and requests are sent directly to Apache;
Varnish: Requests are routed to Varnish running with default parameters. Varnish contacts the always-on back-end Apache server as necessary;
Varnish (Aggressive caching): Requests are routed to Varnish running with ttl and grace period set to one hour, and Varnish contacts the always-on back-end Apache server as necessary;
Varnish (sleep and bot redirection): Requests are routed to Varnish running with aggressive cache settings. Varnish runs the power-control scheme in Section 6.3 and Section 6.2. Wake and sleep times are set to 5 seconds and 5 seconds, respectively, after Varnish makes the request. Is this one current? Bots are redirected if the back-end is asleep when a request arrives;
Varnish (Sleep, bot redirection and survey): Requests are routed to Varnish running with aggressive settings, running the power-control scheme described above. Wake and sleep times are also as above, with bot redirection active. Request from users are redirected to a survey while the server is waking (as described in Section 6.5).

In the last two configurations, the algorithm works with a delay budget of 50 ms, for 90% of requests and uses 5 seconds as an estimate for the turn-around-time.

7.3 Results

First, let us see if our technique can save power in practice. Figure 9 shows the power usage for the five different runs

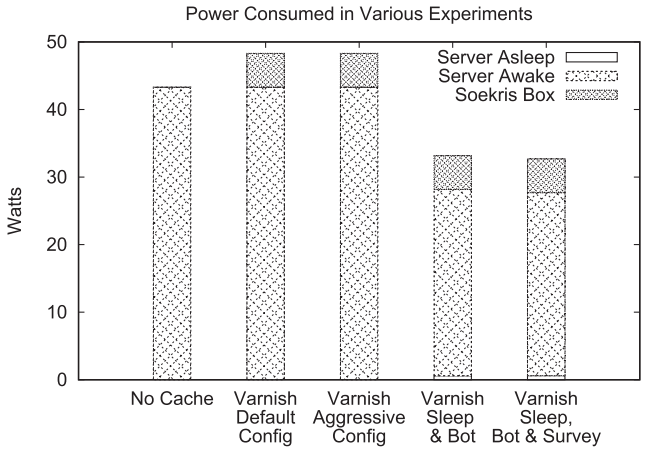

FIGURE 9. Comparison of the results of various experiments in terms of energy used.

described above. The run with Apache alone, and no Varnish gives us a baseline of close to 45 W. The second and third experiments do not attempt to power the server on and off, and so the power usage is simply the sum of power usage of the server and Soekris box. The final two experiments do produce an actual power saving; the extra power used by keeping the Soekris box on is more than matched by the power saved by letting the server sleep. The final bars show that it is possible to save power of ∼25% even accounting for this overhead. Note that the server we used has quite a modest power usage, and greater power savings would be seen with more power-hungry servers.

Now, we need to see the impact of this power saving on performance. Table 3 shows the breakdown of how requests are served. The first column shows the results when Apache fields all requests. While most requests are successful, a number are for files that do not exist, and these are listed under the *404* heading, for File Not Found. We also record the delays involved in serving the files. We see that the mean response time is around a millisecond, and the maximum delay is around 0.4 seconds.

The second column shows the impact of introducing Varnish to the system. Observe that a significant amount of the content, around 7000 requests or 17%, can be cached and served by Varnish. In terms of performance, we see that the low-power reverse proxy cannot serve content as quickly as the Apache server. The low power device only increases average delay by around 5 ms, even though all requests are directed through it. There are a small number of requests with larger delays and we can look at the cumulative distribution of these in Fig. 10. We see that the fraction of requests with a greater than 50 ms is small.

The third column shows the impact of adjusting Varnish's settings to allow more aggressive caching of content. We see that this is quite effective in increasing the number of requests served by Varnish, raising the number to around 12000, or

TABLE 3. Statistics showing results of HTTP requests for each configuration.

Experiment label	No Varnish	Varnish (default config)	Varnish (aggressive config)	Varnish (sleep & bot)	Varnish (sleep, bot & survey)
Requests	40305	40305	40305	40305	41696
— Served By Apache	40298	33288	27673	20280	20245
— Served By Varnish	0	7010	12625	20018	21444
— 200	36846	36837	36834	29879	29922
— 404	3459	3459	3459	2674	2695
— 503	0	9	12	241	163
— malformed requests	7	7	7	7	7
— Bot sleep'd	0	0	0	7511	7525
— People sleep					1391
— Replayed successfully					1041
— Replayed but failed					205
— Replayed but 503'd					14
Mean delay for all requests (s)	0.001	0.007	0.005	0.059	0.009
Mean delay for backend requests (s)	0.001	0.008	0.006	0.112	0.013
Mean delay for varnish (s)		0.002	0.002	0.005	0.005
Max delay apache (s)	0.041	11.238	8.012	12.534	8.053
Max delay varnish (s)		0.207	0.125	3.003	1.984

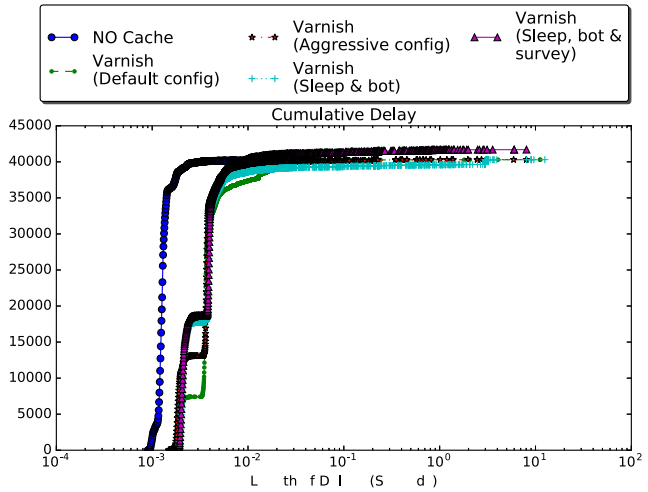

FIGURE 10. Number of delays and their duration across a number of experiments.

about 31%. The mean delays for the requests served by both Varnish and Backend requests does not substantially change, but overall mean delay decreases because the fraction of requests served by Varnish has increased.

The fourth column of the table shows the results when the server is powered on and off, introducing a 5-second delay. The most significant difference we observe here is that requests from spiders while the server is off now receive a Retry After response, indicated by the *Bot Sleep'd* row. There are additional delays in responding to backend requests, due to some requests now having to wait for the server to wake. However, they are still small on average coming to less 60ms. Consulting Fig. 10, we see that we easily meet the target of 90% of requests are served in under 50ms, however there are a cluster of requests that take several seconds to serve corresponding to the time taken to wake the server.

The fifth column of Table 3 shows results where if a request arrives while the backend is sleeping, we suggest the browser take a survey and wake the backend. The request is then replayed after 15 seconds. This results in a higher number of overall requests in this scenario, and we show the results of these extra requests separately in the table. We see introducing the survey reduces the average time to serve requests to less than 10 ms, so it is quite an effective technique. Figure 10 shows the increased number of requests, and that the cluster of requests is now gone, with just a handful of requests taking >1 second to serve, which is broadly in line with the number when we use Varnish but do not put the server to sleep.

7.4 High-performance proxy

In previous sections we reported the results of running our power saving reverse proxy on a net5501, a free-standing low-power single-board computer. This raises the possibility that even though content is being successfully cached during busy periods, the net5501 may not be able to serve content as quickly as a fully-featured server system.

However, the reverse proxy could instead be run on an embedded device designed for high performance networking, such as a IBM® PowerEN™ board [55]. These devices are designed to offer high-performance networking at a lower

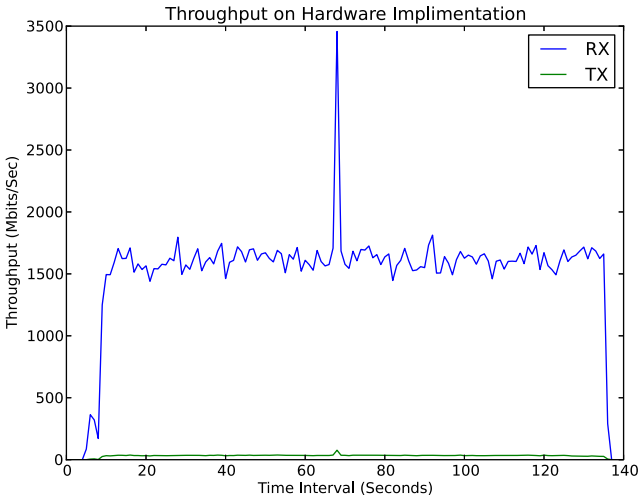

FIGURE 11. Performance of PowerEN™ -based reverse proxy. The peak is due to a mismatch between real time and the rate at which interface statistics are updated.

power consumption than the equivalent PC-based system [56]. In this subsection we demonstrate that using such a device, it is possible to maintain high network throughput. As the device we test is a prototype, we cannot report accurate power usage statistics.

In our test, we ran Varnish on the PowerEN™ board, which proxied content from a server. The content to be proxied was hundreds of files varying from tens of bytes to hundreds of megabytes. Apache JMeter [57, 58] was configured to use multiple threads to request content, with a short ramp-up period. The resulting throughput of the JMeter host is shown in Fig. 11. We see that an embedded device designed for high network performance can maintain a throughput of over 1Gbps, and seems to be able to burst at a higher rate when a number of requests for large files arrive concurrently. This raises the possibility of using an embedded device, even within a server or router, as a reverse proxy to reduce power usage while maintaining high performance.

7.5 Performance on other websites

To evaluate the performance of our scheme on other websites, we built a small simulation tool that reads a web server log file and estimates the power usage of our scheme. The simulation implements the scheme described in Section 6. It does not have a full implementation of Varnish's caching rules, but uses a simplified caching system, including a ttl and grace period. While approximate, when validated against our experiments it gives a power usage within about 1W of that expected for the server's power usage.

As noted in Section 8, many websites actually have a lower load than that used in Section 7. To see how the scheme would perform on such websites, we use the simulator to estimate the performance on two months of log files from three websites: (i) the website of small organization that receives about 130 000 requests in the period; (ii) the website of a small business that receives about 20 000 requests in the period; (iii) a personal website receiving about 7500 requests in the period. For comparison, we expect that the web server used in our evaluation would serve around 2 000 000 requests in this period.

Figure 12 shows the average power usage in each case, estimated by the simulation tool, from the start of the logs. The usage is broken down into the power used by the server while sleeping, the server while awake and the Soekris box running the reverse proxy. Over the full period, the total power usage is around 10 W, 7.5 W and 7 W, respectively, for each of the websites. This indicates that, as we expect, quieter websites could have larger power savings.

8. DISCUSSION

In Section 7 we saw that in practice we could make power savings of 25%. This is a reasonable saving, however it is small compared to the idealized saving for around a 5 -second turn-around-time, of 60–70% (when the power usage of the proxy is factored in). This suggests there may be some scope for improvement, however, it is important to consider that the idealized uses the arrival time of future requests, which must be estimated by a practical scheme.

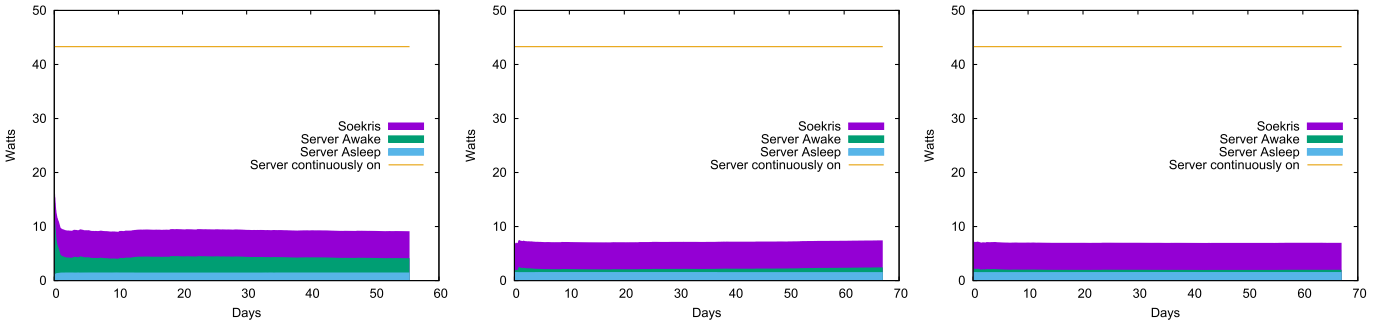

FIGURE 12. Simulated average power usage for three websites under the proposed scheme. On the left, for a small organization, in the middle for a small business and on the right for a personal website.

Our practical power savings do have a performance cost associated with them. For example, the increased baseline delay of a couple of milliseconds that is visible in Fig. 10. Some of this is the inevitable result of introducing an extra proxy system between the client and server, but the exact values relate our experimental setup. For example, we began our experiments with an empty cache with a modest size (128 MB). With a larger cache and a longer test run, it is possible that caching of static content could be more effective again, resulting in fewer wake ups and fewer delays in serving content.

We do not know if our method of introducing a Survey to stall users while the backend is being woken will be irritating for users, however, it has been reported that negative feelings caused by waiting for content can be alleviated can be reduced by explaining such delays [59] or by successful completion of the task at hand [60]. Conducting user validation studies would allow us to see if the added delays or the occasional stalled request may negatively impact on the end user. These studies would require adapting our testbed to serve a selection of different types of web content and will have users rate their experience of accessing this content while the testbed is performing a variety of scenarios, such as simulated heavy request loads and simulated light loads with request stalling. Such a user study would let us gauge our impact on users, but is beyond the scope of the current study.

Taking a more realistic view of the users, requests and web sites might also allow us to improve our power savings. Tracking the behaviour of users could allow us to predict their next likely request. This might be achieved by tracking IP addresses or sessions where active users are monitored. A long period of inactivity or explicit user logout could provide better indications of when to sleep. This could be based on the statistics of the dwell times of typical users, or could be customized for particular website content, possibly anticipating wake-ups from patterns of accesses to cached requests.

Similarly, we could extend the techniques that we use to handle spiders. Many sites seem to experience a high proportion of traffic from spiders [47, 61] and it may be possible to use other features of spiders [62]. Site maps are commonly employed by websites to inform web crawlers, both of the structure of a website, and the frequency of updates to its content. These could be used to help reduce the number of requests by spiders to see if page content has been updated or prevent requests for resources that do not need to be indexed. Another commonly-requested resource is `robots.txt`, a text file defining which well behaved spiders are allowed to crawl the page and what URLs they are allowed or not allowed to follow. Both the `robots.txt` and site map files could be used to mediate the requests from spiders.

Another important question is how to generalize our results to other web traffic or server configurations. Based on a search for sites with online statics publicly available, we find many sites outside the Alexa top 1 000 000 have a lower load than the server we consider here. Other aspects of a site's traffic may have patterns that can be exploited. If request patterns are highly predictable, or such that delays in responses of a few seconds are not a concern, then this could lead to improved power savings. Alternatively, if content is highly dynamic or has tight constraints on how quickly it must be served, then this will make power saving more challenging.

We have considered a situation with a single server hosted on physical hardware that can be powered down. A similar configuration with a higher-powered server should result in the scheme achieving larger savings. The principle of the scheme also generalizes to other configurations. Multiple servers hosted on multiple physical systems could be cached by a single reverse proxy, spreading the cost of a higher-performance proxy over several web servers.

We have noted some challenges with the dependability of WoL, however dependability seemed to be improving and recent network cards that we tested were already at a point where servers could be woken in the region of a hundred times. A more advanced network card might allow the running of a network stack or even whole reverse proxy on the card while asleep. The results in Section 7.4 indicate that smart high-performance network devices, such as IBM® PowerEN™ [55] could be used as reverse proxies.

Hosting of web servers on virtual hardware is, of course, also common. One way to apply the power-saving scheme would be to suspend and resume the virtual hardware. This may result in indirect power savings because of lower resource usage on the physical host, though we expect the savings would be smaller unless enough virtual hosts were suspended to allow a physical machine to sleep. This might be facilitated by allowing migration of suspended VMs to reduce the number of physical hosts currently required.

9. CONCLUSION

We have explored the use of a low-powered reverse proxy to save power by powering off a web server leaving the reverse proxy on. We have considered features of web traffic that facilitate this, including traffic patterns and prevalence of spiders. We demonstrated that if a server is not too busy and can recover from a low-power state quickly (say 5–10 seconds) then it is possible to save power by putting a high-power web server to sleep during low activity periods. In our testbed, we were able to save 25% energy while keeping performance at an acceptable level.

Acknowledgments

This research was supported by HEA PRTLI Cycle 5 TGI and a grant from Science Foundation Ireland and co-funded under the European Regional Development Fund by Grant Number 13/RC/2077. We thank John Walsh for the use of the MSI Server.

REFERENCES

[1] Hilty, L. *et al.* (2009) *The role of ICT in energy consumption and energy efficiency*. https://cordis.europa.eu/project/rcn/87015/factsheet/en.

[2] Gelenbe, E. and Caseau, Y. (2015) The impact of information technology on energy consumption and carbon emissions. *Ubiquity*, 2015, 1.

[3] Gandhi, A. *et al.* (2011) The case for sleep states in servers. *Proc. 4th Workshop on Power-Aware Computing and Systems 2*. ACM, New York, NY, USA.

[4] Xi, S.L. *et al.* (2013) Understanding the critical path in power state transition latencies. *Proc. of the 2013 Int. Symposium on Low Power Electronics and Design*, pp. 317–322. ACM, New York, NY, USA.

[5] O'Dwyer, K. J. *et al.* (2013) Power saving for web servers using proxies. In *Sustainable Internet and ICT for Sustainability (SustainIT)*, 2013, pp. 1–5. IEEE, United States.

[6] Kim, Y.G., Kong, J. and Chung, S.W. (2018) A survey on recent OS-level energy management techniques for mobile processing units. *IEEE Trans. Parall. Distr. Syst.*, 29, 2388–2401.

[7] Chetty, M. *et al.* (2009) It's not easy being green: understanding home computer power management. *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pp. 1033–1042. ACM, New York, NY, USA.

[8] Yu, Y. and Bhatti, S.N. (2014) The cost of virtue: reward as well as feedback are required to reduce user ICT power consumption. *Proc. of the 5th Int. Conf. on Future Energy Systems*, pp. 157–169. ACM, New York, NY, USA.

[9] Perrucci, G.P., Fitzek, F.H. and Widmer, J. (2011) Survey on energy consumption entities on the smartphone platform. *2011 IEEE 73rd Vehicular Technology Conf. (VTC Spring)*, pp. 1–6. IEEE, United States.

[10] Bohrer, P. *et al.* (2002) *The case for power management in web servers. Power Aware Computing*, pp. 261–289. Springer, Boston, MA.

[11] Schönherr, J.H. *et al.* (2010) Event-driven processor power management. *Proc. of the 1st Int. Conf. on Energy-Efficient Computing and Networking, New York, NY, USA e-Energy'10*, pp. 61–70. ACM, New York, NY, USA.

[12] Barroso, L. A. and Hölzle, U. (2007) The case for energy-proportional computing. *Computer*, 12, 33–37.

[13] Pereira, R. *et al.* (2017) Energy efficiency across programming languages. *ACM SIGPLAN Int. Conf. on Software Language Engineering*. ACM, New York, NY, USA.

[14] Chowdhury, S.A., Sapra, V. and Hindle, A. (2016) Client-side energy efficiency of HTTP/2 for web and mobile app developers. *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd Int. Conf. on*, pp. 529–540. IEEE, United States.

[15] Sapra, V. and Hindle, A. (2016) Web servers energy efficiency under HTTP/2. *PeerJ Preprints*, 4, e2027v1.

[16] Mittal, S. and Vetter, J.S. (2015) A survey of methods for analyzing and improving GPU energy efficiency. *ACM Comput. Surv. (CSUR)*, 47, 19.

[17] Srikantaiah, S., Kansal, A. and Zhao, F. (2008) Energy aware consolidation for cloud computing. *Proc. HotPower'08 Proc. of the 2008 Conf. on Power Aware Computing and Systems*. ACM, New York, NY, USA.

[18] Kindelsberger, J., Willnecker, F. and Krcmar, H. (2015) Long-term power demand recording of running mobile applications. *2015 IEEE 10th Int. Conf. on Global Software Engineering Workshops (ICGSEW), July*, 18–22.

[19] Xu, H. and Li, B. (2014) Reducing electricity demand charge for data centers with partial execution. In *Proc. 5th Int. Conf. on Future Energy Systems, New York, NY, USA e-Energy'14*, pp. 51–61. ACM, New York, NY, USA.

[20] Chase, J. *et al.* (2001) Managing energy and server resources in hosting centers. *SIGOPS Oper. Syst. Rev.*, 35, 103–116.

[21] Xu, H. and Li, B. (2013) Anchor: a versatile and efficient framework for resource management in the cloud. *IEEE Trans. Parall. Distr. Syst.,*, 24, 1066–1076.

[22] Lo, D., Cheng, L., Govindaraju, R., Barroso, L.A. and Kozyrakis, C. (2014) Towards energy proportionality for large-scale latency-critical workloads. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pp. 301–312. IEEE.

[23] Kliazovich, D. *et al.* (2012) Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *J. Supercomput.*, 62, 1263–1283.

[24] Calheiros, R.N. *et al.* (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software Pract. Exper.*, 41, 23–50.

[25] Alshammari, D., Singer, J. and Storer, T. (2018) Performance evaluation of cloud computing simulation tools. In *2018 IEEE 3rd Int. Conf. on Cloud Computing and Big Data Analysis (ICCCBDA)* (), pp. 522–526. IEEE, United States.

[26] Ahmad, B. *et al.* (2018) Analysis of energy saving technique in cloudsim using gaming workload. *Cloud Comput.*, 2018, 143.

[27] Atwal, K.S. and Bassiouni, M. (2016) A novel approach for simulation and analysis of cloud data center applications. In *2016 IEEE Int. Conf. on Smart Cloud (SmartCloud)*, pp. 164–169. IEEE, United States.

[28] Mastelic, T. *et al.* (2015) Cloud computing: survey on energy efficiency. *ACM Comput. Surveys*, 47, 33.

[29] Shuja, J. *et al.* (2016) Survey of techniques and architectures for designing energy-efficient data centers. *IEEE Syst. J.*, 10, 507–519.

[30] Shaheen, Q. *et al.* (2018) Towards energy saving in computational clouds: taxonomy, review, and open challenges. *IEEE Access.*, 6, 29407–29418.

[31] Hameed, A.*et al.* (2016) A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 98, 751–774.

[32] Oró, E. *et al.* (2015) Energy efficiency and renewable energy integration in data centres. Strategies and modelling review. *Renew. Sust. Energy Rev.*, 42, 429–445.

[33] You, X. *et al.* (2017) A survey and taxonomy of energy efficiency relevant surveys in cloud-related environments. *IEEE Access.*, 5, 14066–14078.

[34] Bianco, A., Mashayekhi, R. and Meo, M. (2017) On the energy consumption computation in content delivery networks. *Sustain. Comput. Inform. Syst.*, 16, 56–65.

[35] Mathew, V., Sitaraman, R.K. and Shenoy, P. (2012) Energy-aware load balancing in content delivery networks.

In *IEEE INFOCOM, 2012 Proceedings*, pp. 954–962. IEEE.
[36] Araujo, J. INFOCOM, 2012 Proceedings IEEE *et al.* (2015) Energy efficient content distribution. *Comput. J.*, 59, 192–207.
[37] Mathew, V., Sitaraman, R.K. and Shenoy, P. (2015) Energy-efficient content delivery networks using cluster shutdown. *Sustain. Comput. Inform. Syst.*, 6, 58–68.
[38] Fang, C. *et al.* (2017) Distributed energy consumption management in green content-centric networks via dual decomposition. *IEEE Syst. J.*, 11, 625–636.
[39] Braun, H. and K. Claffy (1994) Web traffic characterization: an assessment of the impact of caching documents from the NCSA's web server. In Second International World Wide Web (WWW) Conference'94 (Oct.), Chicago, IL. Elsevier Science Publishers B. V. Amsterdam, The Netherlands.
[40] Pitkow, J. (1999) Summary of WWW characterizations. *World Wide Web*, 2, 3–13.
[41] Barford, P. *et al.* (1999) Changes in web client access patterns—characteristics and caching implications. *World Wide Web*, 2, 15–28.
[42] Bent, L. *et al.* (2004) Characterization of a large web site population with implications for content delivery. In *Proc. of the 13th Int. Conf. on World Wide Web, New York, NY, USA WWW'04*, pp. 522–533. ACM, New York, NY, USA.
[43] Cao, J. *et al.* (2004) Stochastic models for generating synthetic HTTP source traffic. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, March* (Vol. 3), pp. 1546–1557. IEEE, United States.
[44] Ihm, S. and Pai, V.S. (2011) Towards understanding modern web traffic. In *Proc. of the 2011 ACM SIGCOMM Conf. on Internet Measurement Conf.*, pp. 295–312. ACM.
[45] Padmanabhan, V.N. and Qiu, L. (2000) The content and access dynamics of a busy web site: findings and implications. *ACM SIGCOMM Comput. Commun. Rev.*, 30, 111–123.
[46] Kamp, P.-H. (2006–2019). Varnish cache. [Version: varnish-3.0.1 revision 6152bf7].
[47] Gaffan, M. (2012). *What Google doesn't show you: 31% of website traffic can harm your business*. http://www.incapsula.com/the-incapsula-blog/item/225-what-google-doesnt-show-you-31-of-website-traffic-can-harm-your-business.
[48] Olston, C. and Najork, M. (2010) Web crawling. *Found. Trends Inf. Ret.*, 4, 175–246.
[49] Kumar, M., Bhatia, R. and Rattan, D. (2017) A survey of web crawlers for information retrieval. *Wires Data Min. Knowl. Discovery*, 7, e1218.
[50] Fielding, R. *et al.* (1999). RFC 2616: hypertext transfer protocol — HTTP/1.1.
[51] Jerkovic, J. (2009) *SEO Warrior: essential techniques for Increasing Web Visibility* (). O'Reilly Media.
[52] Mostowfi, M. (2018) HTTP timed redirection to reduce the energy use of servers. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conf. (CCWC)*, pp. 496–501. IEEE, United States.
[53] UEFI Forum (1996–2019). Advanced configuration and power interface. http://www.acpi.info.
[54] Menarini, M. *et al.* (2013) Green web services: improving energy efficiency in data centers via workload predictions. In *Proc. of the 2nd Int. Workshop on Green and Sustainable Software* (Vol. 13), pp. 8–15. IEEE Press, Piscataway, NJ, USA GREENS.
[55] Golander, A. *et al.* (2010) IBM's PowerEN developer cloud: fertile ground for academic research. In *IEEE 26th Convention of Electrical and Electronics Engineers in Israel (IEEEi)*, pp. 803–807.
[56] Heil, T. *et al.* (2014) Architecture and performance of the hardware accelerators in IBM's PowerEN processor. *ACM Trans. Parall. Comput.*, 1, 5.
[57] Halili, E. (2008) *Apache JMeter: A Practical Beginner's Guide to Automated Testing and Performance Measurement for Your Websites*. Packt Pub Limited, Birmingham, United Kingdom.
[58] Matam, S. and Jain, J. (2017) *Pro Apache JMeter: web application performance testing*. Apress, New York, NY, USA.
[59] Ryan, G. and Valverde, M. (2006) Waiting in line for online services: a qualitative study of the user's perspective. *Inf. Syst. J.*, 16, 181–211.
[60] Ryan, G., del Mar Pàmies, M. and Valverde, M. (2015) WWW= wait, wait, wait: emotional reactions to waiting on the internet. *J. Electron. Commer. Res.*, 16, 261.
[61] Doran, D., Morillo, K. and Gokhale, S.S. (2013) A comparison of web robot and human requests. In *Proc. of the 2013 IEEE/ACM Int. Conf. on Advances in Social Networks Analysis and Mining*, pp. 1374–1380. ACM, New York, NY, USA.
[62] Brown, K. and Doran, D. (2018) Contrasting web robot and human behaviors with network models, Journal of Communications 13(8), January 2018, doi: 10.12720/jcm.13.8.473-481.