GreenCache: Augmenting Off-the-Grid Cellular Towers with Multimedia Caches

Navin Sharma, Dilip Kumar Krishnappa, David Irwin, Michael Zink, and Prashant Shenoy
University of Massachusetts Amherst
{nksharma,shenoy}@cs.umass.edu, {krishnappa,irwin,zink}@ecs.umass.edu

ABSTRACT

The growth of smartphones combined with advances in mobile networking have revolutionized the way people consume multimedia data. In particular, users in developing countries primarily rely on smartphones since they often do not have access to more powerful (and more expensive) computing devices. Unfortunately, cellular networks in developing countries have historically had low reliability, due to grid instability and lack of infrastructure. The situation has led network operators to experiment with running cellular towers "off the grid" using intermittent renewable energy sources. In parallel, network operators are also experimenting with co-locating server caches close to cell towers to reduce access latency and backhaul bandwidth. In this paper, we study techniques for optimizing multimedia caches for intermittent renewable energy sources. Specifically, we examine how to apply a blinking abstraction proposed in prior work, which rapidly transitions servers between an active and inactive state, to improve the performance of a multimedia cache powered by renewables, called GreenCache. Our results show that GreenCache's staggered load-proportional blinking policy, which coordinates when servers are active over brief intervals, results in 3X less buffering (or pause) time by the client compared to an activation blinking policy, which simply activates and deactivates servers over long periods as power fluctuates, for realistic power variations from renewable energy sources.

Categories and Subject Descriptors

C.5.0 [**Computer System Implementation**]: General

General Terms

Design, Management, Performance

Keywords

Cellular Tower, Intermittent, Renewable Energy, Cache

1. INTRODUCTION

The growth of smartphones combined with advances in mobile networking have revolutionized the way people consume multimedia data, e.g., video and audio [1]. For example, a recent study indicates that YouTube users now watch more than 20% of YouTube videos from their smartphones [3]. The fraction of users in developing countries that rely on smartphones to access videos is even higher, since these users often do not have access to more powerful (and more expensive) computing devices, such as desktops, laptops, and tablets. Unfortunately, cellular networks in developing countries have historically had low reliability, since cellular towers typically run off power from the electric grid. As the recent Indian blackout in July 2012, which left 700 million people without power for two days, indicates, the grid in these countries is often unstable.

In some cases, connections to an even unstable electric grid may not be available. For example, in India, 150,000 out of 400,000 cell towers do not have reliable access to the electric grid [7]. Grid instability and lack of infrastructure has led network operators to run cellular towers "off the grid" [4]. Today's "off the grid" cellular towers operate off diesel generators that are costly to maintain, in large part, because operators must continuously refuel them with expensive and "dirty" diesel fuel. Since i) network operators view developing countries as a large potential market and ii) many potential users in these countries reside in rural areas without access to a reliable electric grid, there is a strong financial incentive to deploy off-the-grid cell towers powered by renewable energy sources, e.g., from either solar panels or wind turbines.

In parallel, the growth of smartphones as a primary end-point for multimedia data has led to a significant rise in bandwidth usage of cell towers, which requires higher backhaul bandwidth to fetch data, as well as additional spectrum or denser tower deployments to deliver this data. As a result, in addition to the problems above, network operators have also sought ways to reduce the bandwidth consumption of multimedia applications. Traditionally, network operators have deployed caches only at centralized locations, such as operator peering points, in part, for both simplicity and ease of management [16]. However, researchers and startups have recently recognized the benefit of placing caches closer to edge [16, 2]. Co-locating server caches closer to cell towers would both reduce access latency, by eliminating the need to route requests for cached data through a distant peering point, and reduce backhaul bandwidth usage from the cell tower to the peering point. Caches co-located with cell towers primarily target multimedia data, since it consumes the largest fraction of bandwidth and is latency-sensitive.

In this paper, we consider how to operate a distributed cache for multimedia data that is co-located with a cell tower powered using renewable energy. A key challenge in using renewable sources such as solar and wind is that they are intermittent and the amount of power they generate fluctuates depending on environmental conditions. Consequently we assume that when renewable generation is plentiful, e.g., on sunny or windy days, the generated power is sufficient to power both the cell tower and the co-located servers that house the multimedia cache. However during periods of scarcity, the renewable sources (combined with a modest-sized battery) may

[1]This research is supported by NSF grants CNS-1117221, CNS-0916577, EEC-0313747, CNS-0855128, and CNS-0834243.

not generate sufficient power and the setup must reduce its total energy footprint. In this case, we assume that the available scarce power is used to keep the cell tower up and running (so that consumers do not lose cellular service) and that there may not be enough energy to always operate the multimedia cache.

Recently we proposed a new abstraction, called blinking, to adapt server clusters to intermittent power. Blinking [11] dynamically regulates the energy footprint of servers, to match available power, by rapidly, e.g., once a minute, transitioning servers between a high-power active state and a low-power inactive state. We showed that blinking improves the performance of an in-memory cache for small objects, e.g., Memcached storing simple strings and binary values. Caches for multimedia data differ in important ways. First, multimedia contents are often much larger in size than Memcached objects, which are limited to 1MB in size, and thus require persistent storage for caching. Second, unlike Memcached objects, multimedia data can be streamed, such that a video need not be completely transmitted to a client before the client can display it. Instead, the client can start playing a video as soon as it receives data for the first few seconds, as long as the client continues streaming data from the server while the video is playing. This characteristic is well-suited to the blinking abstraction, since different chunks of a video can be stored on different cache servers in a cluster. The challenge is to activate nodes in a way that segments of the video are streamed to the client in a sequence that allows for uninterrupted video playback.

In this paper, we propose GreenCache, a distributed cache for multimedia data co-located with cell towers that run off renewable energy. GreenCache leverages the blinking abstraction to significantly (i) modulate its energy footprint to match available power, (ii) reduce bandwidth usage from the cell tower to backend servers, and (iii) reduce access latency for clients despite fluctuations in available power. As discussed above, minimizing bandwidth usage (or cost) and maximizing users' experience, e.g., by reducing buffering time, are two primary goals of a multimedia cache. We analyze video traffic behavior of a large number of users for the most popular user-generated video site, YouTube, and exploit traffic characteristics and video properties to design new placement and blinking policies for minimizing bandwidth usage and maximizing users' experience. In achieving this goal, our work makes the following contributions:

YouTube Trace Analysis. To motivate the design and feasibility of GreenCache and the use of the blinking abstraction in streaming environments, we collect YouTube video traces from the University of Connecticut for 3 days, and analyze them to find important characteristics about viewers behavior and the popularity of YouTube videos.

GreenCache Design. We detail the design of a blink-aware distributed multimedia cache and its advantages over existing multimedia caches when using renewable energy sources. Our design uses an always-active proxy to receive requests, while masking blinking, from mobile clients.

Buffering Time Reduction Techniques. We detail techniques for reducing video buffering time at the client. Our approach combines a load-proportional data layout with a staggered load-proportional blinking policy to minimize per video average buffering time. Our design also uses a popularity-aware cache eviction policy to minimize bandwidth usage from cell towers to backend servers.

GreenCache Implementation and Evaluation. We implement GreenCache on a laboratory prototype comprising a real 4G WiMAX base-station we have deployed on the UMass campus and a cluster of ten Mac Minis that can be powered using either a renewable source or a programmable power supply that mimics these renewables. We then evaluate our cache for bandwidth usage and buffering time at different (fixed and oscillating) power levels using our WiMAX base-station and WiMAX clients. Our experimental workloads are based on user requests from the real network traces mentioned above. Our results show that a staggered load-proportional blinking policy, which staggers when nodes are active and varies the length of the active interval, results in 3X less buffering (or pause) time by the client compared to an activation blinking policy, which simply activates and deactivates nodes as power fluctuates, for realistic power variations from renewable energy sources.

The remainder of the paper is outlined as follows. Section 2 describes the benefits of blinking for caches running on intermittent power. Section 3 analyzes YouTube video traces from the University of Connecticut, while Section 4 presents design techniques for a blinking multimedia cache. Section 5 details our implementation of a small distributed cache prototype, and Section 6 then evaluates our prototype for two metrics – bandwidth cost and buffering time. Finally, Section 7 discusses related work and Section 8 concludes.

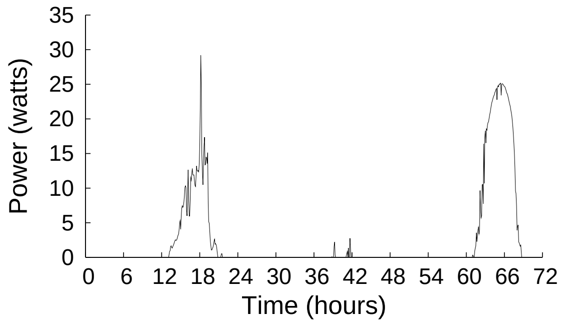
2. CACHE AND INTERMITTENT POWER

Our work assumes cellular towers that are powered using off-the-grid sources such as solar and wind energy. We assume that the cellular network employs in-network multimedia caches to deliver popular content to mobile end-users via cellular data connections. To reduce latency, backend server load, as well as backhaul/upstream bandwidth, we assume that these caches are located close to, or at, the cellular towers of 3G/4G base stations. As noted in Section 1, while today's legacy cellular networks are not able to co-locate caches very close to cell towers due to inherent architectural limitations, research efforts and startup companies [16, 2] are developing techniques and products to address these limitations; thus we envision that future cellular networks will be able to deploy computation and storage, e.g., server caches, near base stations—similar to what we assume here.

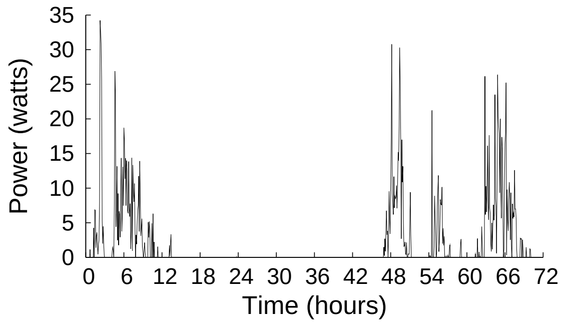
The use of caches in a cellular network has many benefits. Since the popularity distribution of videos is often heavy-tailed, i.e., a small set of videos out of the pool of all available videos are significantly more popular than the rest, a well-designed cache cluster can reduce the back-end traffic, and thus the bandwidth cost on the uplink. Our earlier work analyzed the benefits of caching for YouTube videos through trace-based simulations, and showed that caching can indeed reduce uplink bandwidth [17, 8]. Further, an increase in storage capacity increases the efficiency of the cache. i.e., the larger the cache's storage the higher the potential that a requested video can be served from the cache.

However, co-locating caches near cellular towers also raises challenges. First, the presence of servers and storage near the cellular tower increases the energy footprint of the tower. The problem is exacerbated in developing countries with an unreliable grid. In off-the-grid towers with renewable sources, we must deal with the additional problem intermittency in these energy sources. Figure 1 shows how both solar and wind energy can vary each day. The figure shows that even on generally sunny or windy days the output from renewables can fluctuate significantly. Even with the use of batteries, there may not be sufficient energy to operate the base station and the servers during periods of energy scarcity (e.g., on cloudy days with low solar output). We assume that the server caches must somehow reduce their energy usage during such periods while the base station stays up—to the extent possible—to provide cellular service to end users.

To handle energy scarcity, we assume a cache architecture that comprises of a number of low-power servers, since a single large



(a) Solar



(b) Wind

Figure 1: Solar and wind energy harvesting from our solar panel and wind turbine deployment on three consecutive days in Sep 2009.

cache is not well-suited to operating off intermittent renewable energy sources. To understand why, consider that since computing equipment, including servers and network switches, is not energy-proportional, it is not possible to scale down performance with power usage to adapt to changes in available power. For example, a 300W server may have a dynamic power range when active between 200W and 300W; thus, if power generation drops below 200W the server must be shutdown. To mimic energy-proportionality, an alternative approach uses smaller, lower-power servers, which can each be activated and deactivated to match available power [14]. The advantage of this approach is that it allows the cache size to scale up and down based on available power. However, it introduces a new complication: if servers are inactive due to power shortages by renewables then the data cached on them becomes unavailable. If data resides on an inactive server, the client must either wait until the server is active, e.g., there is enough power, or retrieve the already cached data again from the origin server.

With a distributed cache, there are two ways to reduce energy use during shortfall periods. First, some or all caches can be temporarily powered down, but doing so implies that users will not see any benefits of caching in these periods. A better approach is to use blinking [11] where each node rapidly transitions (duty cycles) between sleep and awake modes. Blinking allows caches to provide service, albeit at degraded performance, during shortfall periods. Our earlier work demonstrated the feasibility of implementing blinking on commodity hardware—the rate and duration of blinking can be adjusted to match the energy availability. Longer sleeps can be used, for instance, during very low energy supply periods.

In essence, blinking provides a cache with new options in its design space. Rather than having a small cache composed of the number of always-on servers the available power can sustain, blinking provides the option of having a much larger cache composed of servers that are active for only a fraction of time each blink interval, e.g., active for 10 seconds during each minute interval. The use of blinking raises new challenges in multimedia cache design. The main challenge is to ensure smooth uninterrupted video playback even while blinking. Doing so implies that caches have to stream additional data during their active periods to compensate for lack of network streaming during sleep periods. Further, end-clients will need to employ additional client-side buffers and might see higher startup latencies.

Since multimedia applications are very sensitive to fluctuation in network bandwidth that might cause delayed data delivery at the client, most applications like video players employ a buffer to smooth out such fluctuations and provide an uninterrupted, error free play out of the data. This buffer, which already exists for most

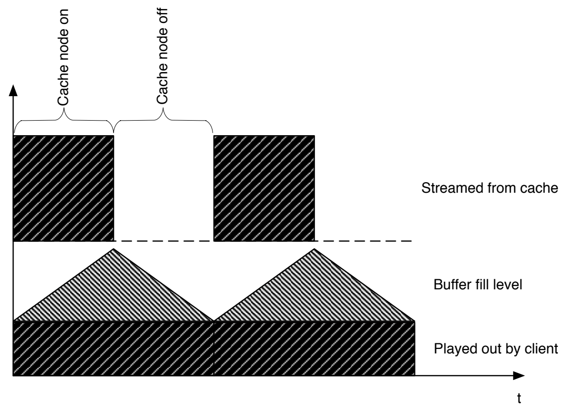


Figure 2: The top part of the figure shows a potential streaming schedule for a blinking node while the bottom half shows the smooth play out with is achieved with the aid of a client-side buffer.

multimedia applications on the client side, integrates well into the blinking approach since it also allows the cache to bridge outage times in individual cache servers, as shown in Figure 2. A blinking cache will stream additional chunks when active, which are buffered at the client. As shown in this figure, the player is then able to play the video smoothly and masks interruptions from the viewer as long as it gets the next chunk of data before the previous chunk has finished playing.

Finally, in a typical cell tower or 3G/4G/WiMAX scenario the downstream bandwidth ($\sim$30-40 Mbps) is much less than the bandwidth a cache server can provide, which is generally limited by its network card and disk I/O. So, the cache server can potentially reduce its energy consumption by sending data at its full capacity for a fraction of a time interval (usually few seconds) and going to a low-power state for the remaining period of the time interval, as shown in Figure 2. In essence, the server could employ the blinking abstraction to reduce its energy footprint while still satisfying the downstream bandwidth requirement of the cell tower or WiMAX station. Moreover, blinking facilitates a cache to employ more servers than it can keep active with the available power, and thus provides an opportunity to reduce server load and bandwidth usage.

The primary drawback of a blinking cache is that it stalls a request if the requested video is not currently available on an active server. If a client requests a video that is present on an inactive server, the cache can either get the video from the back-end server or the client pauses play out until the inactive server becomes active. While getting the video from the back-end server, instead of

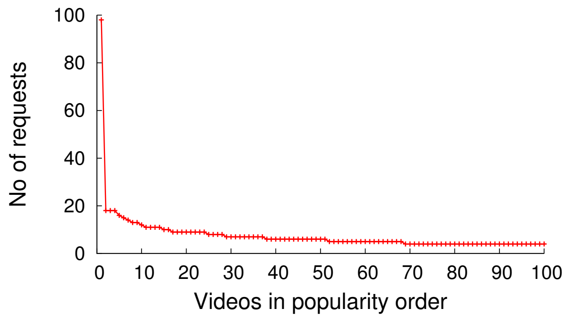


Figure 3: Video Popularity (100 out of 105339)

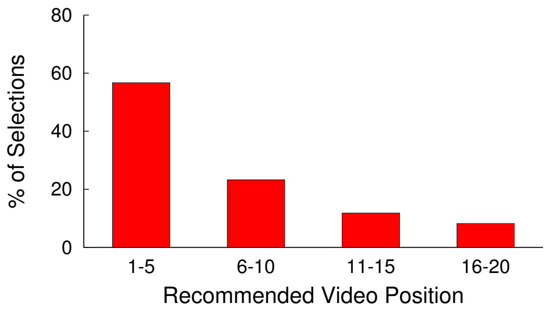


Figure 4: Related Video Position Analysis

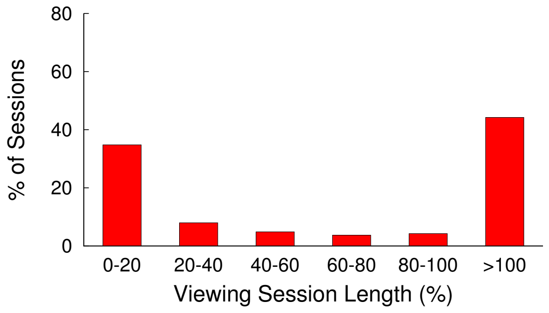


Figure 5: Video Switching Time Analysis

waiting for the inactive server to become active, reduces the buffering time, it increases the bandwidth cost. As described in Section 4, GreenCache uses a low-power always-on proxy and staggered load-proportional blinking policy to reduce buffering time while sending requests to back-end servers only if data is not available in the cache.

3. GREENCACHE FEASIBILITY: TRACE ANALYSIS

To inform the design of GreenCache based on the characteristics of multimedia traffic and viewer behavior, we analyze a network trace that was obtained by monitoring YouTube traffic entering and leaving a campus network at the University of Connecticut. We believe that such a trace can be seen as a first order approximation for a community that is served by a WiMAX or 3G/4G base station. (One can easily imagine that a University campus or a substantial part of the campus could be served by such a base station.) Indeed, some Universities have started installing WiMAX base stations on their campuses as part of their network infrastructure [9].

The network trace is collected with the aid of a monitoring device consisting of PC with a Data Acquisition and Generation (DAG) card [1], which can capture Ethernet frames. The device is located at a campus network gateway, which allows it to capture all traffic to and from the campus network. It was configured to capture a fixed length header of all HTTP packets going to and coming from the YouTube domain. The monitoring period for this trace was 72 hours. This trace contains a total of 105,339 requests for YouTube videos out of which $\sim 80\%$ of the video requests are single requests which leaves about 20% of the multiple requests to take advantage of caching of the requested videos. We would like to point out that a similar caching potential (24% in this case) has been reported in a more global study of 3G networks traffic analysis by Erman et al. [6].

Figure 3 shows the popularity distribution of the 100 most popular videos, which is obtained based on the user requests recorded in the trace. This figure only shows the 100 most popular videos since the trace contains many videos with a very low popularity ($<$ 10 requests) and we wanted to depict the distribution of the most popular videos in more detail. The data obtained from the analysis of the trace shows that, despite the very long tail popularity distribution, caching can have an impact on the performance of such a video distribution system.

In earlier work [8], we have shown that, not only caching but also the prefetching of prefixes of videos that are shown on the related video list of a YouTube video page can improve the viewers experience of watching videos. Further analysis of the trace revealed that 47,986 request out of the 105,339 had the tag **related_video** ($\sim 45\%$), which indicates that these videos have been chosen by viewers from the related video list that is shown on each YouTube video's web page. In addition to identifying videos that are selected from the related list, we also determine the position on the related list the video was selected from and show the result in Figure 4. It shows that users tend to request from the top 10 videos shown on the related list of a video, which accounts for 80% of the related video requests in the trace. This data shows that, prefetching the prefixes of the top 10 videos shown on the related list of a currently watched video can significantly increase viewer's experience, since the initial part can be streamed immediately from a location close to the client. Based on these results, we decided to evaluate a blinking multimedia cache that performs both, traditional caching, and prefix prefetching for the top 10 videos on the related video list.

We also analyze the trace to investigate if viewers switch to a new video before they completely finish watching the current video. In order to analyze this behaviour, we look into the timestamps of a user requesting two consecutive videos. We calculate the difference of these timestamps and compare it with the total length of the first video requested to determine if the user has switched between videos before the previous video is completely viewed.

Figure 5 shows the number of occurrences (in percent out of the total number of videos watched) a video is watched for $x\%$ of its total length. This result shows that only in 45% of the cases videos are watched completely (also this number is similar to the global study performed by Erman et al. [6]). In all other cases only part of the video is watched, with the majority of these cases ($\sim 40\%$) falling in the 0 - 20% viewing session length. This result let us to the decision to divide a video into equal-sized chunks, which allows for the storage of different chunks that belong to a single video on different nodes of the cache cluster. In Section 4.1, we describe how the chunk size is determined and how chunking a video can reduce the uplink bandwidth usage if used on a blinking multimedia cache cluster.

4. GREENCACHE DESIGN

Figure 6 depicts GreenCache's architecture, which consists of a proxy and several cache servers. The proxy maintains a

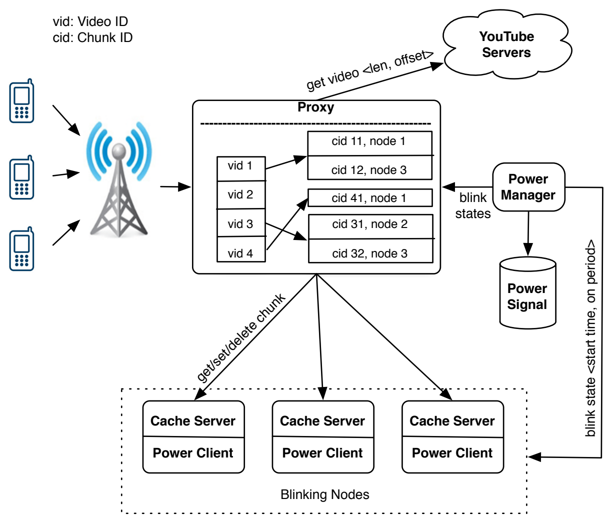


Figure 6: GreenCache Architecture.

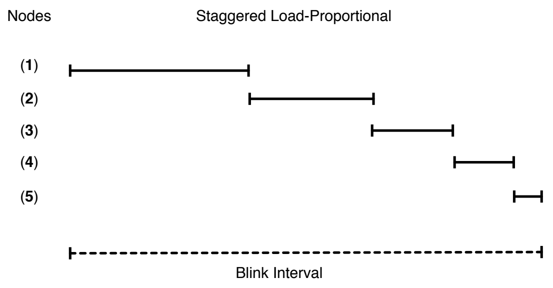


Figure 7: Staggered load-proportional blinking.

video→chunk mapping and a chunk→node mapping, while also controlling chunk server placement and eviction. Clients, e.g., web browsers on smartphones, connect to video servers through the proxy, which fetches the requisite data from one or more of its cache servers, if the data is resident in the cache. If the data is not resident, the proxy forwards the request to the host, i.e., back-end server. The proxy stores metadata to access the cache in its own memory, while video chunks reside on stable storage on each cache server.

GreenCache also includes a power manager, that monitors available power and energy stored in a battery using hardware sensors, e.g., a voltage logger and current transducer. The power manager implements various blinking policies to control nodes' active and inactive intervals to match the cache's power usage to the available power. The power manager communicates with a power client running on each cache server to set the start time and active period every blink interval. The power client activates the node at the start time and deactivates the node after the active period every blink interval, and thus controls node-level power usage by transitioning the node between active and inactive states.

As discussed earlier, the primary objective of multimedia cache is to reduce buffering (or pause) time at the client and the bandwidth usage between the cache and the origin server. Next, we describe GreenCache's techniques to both reduce bandwidth usage to the backend origin server, while also minimizing buffering (or pause) time at the client.

4.1 Minimizing Bandwidth Cost

As Figure 3 indicates, all videos are not equally popular. Instead, a small number of videos exhibit a significantly higher popularity than others. Similar to other multimedia caches, GreenCache has limited storage capacity, requiring it to evict older videos to cache new videos. An eviction strategy that minimizes the bandwidth usage each interval will evict the least popular videos during the next interval. However, such a strategy is only possible if the cache knows the popularity of each video in advance. To approximate a video's future popularity, GreenCache maintains each video's popularity as an exponentially-weighted moving average of a video's accesses, updated every blink interval. The cache then evicts the least popular videos if it requires space to store new videos.

As shown in Figure 5, most videos are not watched completely most of the time. In fact, the figure shows that users of YouTube watch less than 45% of the videos to completion. In addition, users might watch the last half of a popular video less often than the first half of an unpopular video. To account for discrepancies in the popularity of different segments of a video, GreenCache divides a video into multiple chunks, where each chunk's playtime is equal in length to the blink interval. Similar to entire videos, GreenCache tracks chunk-level popularity as an exponentially weighted moving average of a chunk's accesses. Formally, we can express the popularity of the i_{th} chunk after the t_{th} interval as:

$$Popularity_i{}^{t} = \alpha A_i{}^{t} + (1-\alpha) Popularity_i{}^{t-1} \qquad (1)$$

$A_i{}^t$ represents the total number of accesses of the i_{th} chunk in the t_{th} interval, and α is a configurable parameter that weights the impact of past accesses. Further, GreenCache manages videos at the chunk level, and evicts least popular chunks, from potentially different videos, to store a new chunk. As a result, GreenCache does not need to request chunks from the backend origin servers if the chunk is cached at one or more cache servers.

4.2 Reducing Buffering Time

As discussed earlier, blinking increases buffering time up to a blink interval, if the requested chunk is not present on an active server. The proxy could mask the buffering time from a client if the client receives a chunk before it has finished playing the previous chunk. Assuming sufficient energy and bandwidth, the proxy can get a cached chunk from a cache server within a blink interval, since all servers become active for a period during each blink interval. As a result, a user will not experience pauses or buffering while watching a video in sequence, since the proxy has enough time to send subsequent chunks (after the first chunk) either from the cache or the origin server before the previous chunk finishes playing, e.g., within a blink interval. However, the initial buffering time for the first chunk could be as long as an entire blink interval, since a request could arrive just after the cache server storing the first chunk becomes inactive. Thus, to reduce the initial buffering time for a video, the proxy replicates the first chunk of cached videos on all cache servers. However, replication alone does not reduce the buffering time if all servers blink synchronously, i.e., become active at the same time every blink interval. As a result, as discussed next, GreenCache employs a staggered load-proportional blinking policy to maximize the probability of at least one cache server being active at any power level.

4.2.1 *Staggered Load-Proportional Blinking*

As discussed above, we replicate the first chunk of each cached video on all cache servers in order to reduce initial buffering time. To minimize the overlap in node active intervals and maximize the probability of at least one active node at all power levels, Green-

Cache staggers start times of all nodes across each blink interval. Thus, every blink interval, e.g., 60 seconds, each server is active for a different period of time, as well as a different duration (discussed below). At any instant, a different set of servers (and their cached data) is available for clients. Since at low power the proxy might not be able to buffer all subsequent chunks from blinking nodes, clients might face delays or buffering while watching videos (after initially starting them).

To reduce the intermediate buffering for popular videos, GreenCache also groups popular chunks together and assigns more power to nodes storing popular chunks than nodes storing unpopular chunks. Thus, nodes storing popular chunks are active for a longer duration each blink interval. GreenCache ranks all servers from 1...N, with 1 being the most popular and N being the least popular node. The proxy monitors chunk popularity and migrates chunks to servers in rank order. Furthermore, the proxy distributes the available power to nodes in proportion to the aggregate popularity of their chunks. Formally, active period for the i_{th} node, assuming negligible power for inactive state, could be expressed as

$$Active_i = \frac{BI * P * Popularity_i}{MP * \sum_{k=1}^{n} Popularity_i} \qquad (2)$$

BI represents the length of a blink interval, $Popularity_i$ represents the aggregate popularity of all chunks mapped on the i_{th} node, P denotes the available power, and MP is the maximum power required by an active node. Additionally, start times of nodes are staggered in a way that minimizes the unavailability of first chunks, i.e., minimizes the period when none of the nodes are active, every blink interval. Figure 7 depicts an example of staggered load-proportional blinking for five nodes. Note that since the staggered load-proportional policy assigns active intervals in proportion to servers' popularity, it does not create an unbalanced load on the cache servers.

4.2.2 Prefetching Recommended Videos

Most popular video sites display a recommended list of videos to users. For instance, YouTube recommends a list of twenty videos which generally depends on the current video being watched, the user's location, and other factors including past viewing history. From the trace analysis provided in Section 3, we can infer that, users tend to select the next video from recommended videos $\sim$45% of the time. In addition, a user selects a video at the top more often than a video further down in the recommended list. In fact, Figure 4 shows that nearly 55% of the time a user selects the next video from top five videos in the recommended list. To further reduce initial buffering time the proxy prefetches the first chunk of top five videos in the recommended list, if these chunks are not already present in the cache. The proxy fetches subsequent chunks of the video when the user requests the video next.

5. GREENCACHE IMPLEMENTATION

We implement a GreenCache prototype in Java, including a proxy ($\sim$1500 LOC), cache server ($\sim$500 LOC), power manager ($\sim$200 LOC), and power client ($\sim$150 LOC). Mobile clients connect to the Internet through a wireless base station, such as a cell tower or WiMAX base station, which is configured to route all multimedia requests to the proxy. While the power manager and proxy are functionally separate and communicate via well-defined APIs, our prototype run both modules on the same node. The power manager exposes APIs to access the available energy, blink interval, and node's blink state – start time and active period. Our prototype does not require any modification in the base station or mobile clients.

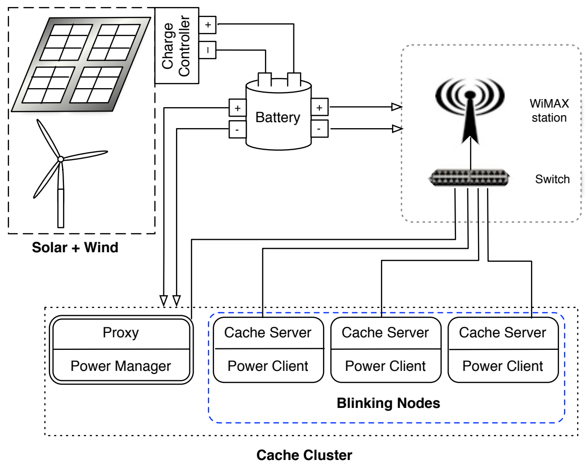


Figure 8: Hardware Prototype.

Both cache server and power client run together on each blinking node.

Our prototype includes a full implementation of GreenCache, including the staggered load-proportional blinking policy, load-proportional chunk layout, prefetching, video chunking, chunk eviction and chunk migration. The proxy uses a Java Hashtable to map videos to chunks and their locations, e.g., via their IP address, and maintains their status, e.g., present or evicted. Since our prototype has a modular implementation, we are able to experiment with other blinking policies and chunk layouts. We implement the activation and proportional policies from the original Blink work [11] to compare with GreenCache's staggered load-proportional policy. The original work applied blinking only to a more general distributed memory cache for small objects (Memcached), and thus had no need for replicating objects based on their popularity. We also implement a randomized chunk layout and the Least Recently Used (LRU) cache eviction policy to compare with the proposed load-proportional layout and popularity based eviction policy, respectively.

Hardware Prototype. We construct a small-scale hardware prototype that uses intermittent power to experiment with GreenCache in a realistic setting. Our current prototype builds off our prototype from Blink [11]. However, unlike that prototype, which uses OLPC nodes, our GreenCache prototype uses more powerful, but energy-efficient, Mac minis. We use a small cluster of ten Mac minis running Linux kernel 2.6.38 with 2.4 GHz Intel Core 2 Duo processors and 2GB of RAM connected together using an energy-efficient switch (Netgear GS116) that consumes 15W. Each Mac mini uses a flash-based SSD with a 40GB capacity. We use one Mac mini to run the proxy and power manager, whereas we run a cache server and power client on other Mac minis. The proxy connects to a WiMAX base station (NEC Rel.1 802.16eBS) through the switch. We use a Linux laptop with a Teletonika USB WiMAX modem to run as a client. We also use a separate server to emulate multiple WiMAX clients. Our emulator limits the wireless bandwidth, in the same way as observed by the WiMAX card, and plays the YouTube trace described below. The WiMAX base station is operational and located on the roof of a tall building on the UMass campus. However, the station is currently dedicated for research purposes and is not open to the general public.

Similar to [11], we use ACPI's S3 suspend-to-ram state as the inactive state. We boot each Mac mini in text mode and unload all unnecessary drivers in order to minimize the time it takes to transition

Buffering time (s)	*Power (%)*				
⇓	***20***	***40***	***60***	***80***	***100***
Blink interval = 30 sec					
Std Dev	7.88	5.33	3	0.52	0.03
90^{th}per	21.25	15.25	9.25	2.25	0.23
Avg.	10.99	6.12	3.88	2.35	0.25
Blink interval = 60 sec					
Std Dev	14.59	10.01	6.79	2.55	0.03
90^{th}per	41.25	28.25	19.25	7.45	0.23
Avg.	20.58	10.19	6.94	3.36	0.25
Blink interval = 90 sec					
Std Dev	24.79	16.69	10.06	3.16	0.03
90^{th}per	66.25	43.25	25.65	4.25	0.23
Avg.	29.44	15.12	8.50	3.22	0.25
Blink interval = 120 sec					
Std Dev	30.52	22.21	13.13	5.29	0.03
90^{th}per	78.25	59.45	31.45	14.45	0.23
Avg.	32.73	21.58	9.81	4.58	0.25

Table 1: Standard deviation, 90th percentile, and average buffering time at different power levels and blink intervals.

into S3. With the optimizations, the time to transition to and from ACPI's S3 state on the Mac mini is one second. Note that much faster sleep transition times, as low as a few milliseconds, are possible, and would further improve GreenCache's performance. We select a blink interval of 60 secs, resulting in a transition overhead of $1/60 = 1.66\%$ every blink interval. With the optimizations above, the power consumption of the Mac mini in S3 and S0 is 1W and 25W respectively. Since GreenCache requires one node, running the proxy and power manager, the switch, and WiMAX base station to be active all the time, its minimum power consumption is 46 W, or 17% of its maximum power consumption.

We power the cluster from a battery that connects to four ExTech 382280 programmable power supplies, each capable of producing 80W, that replay the variable power traces described below. To prevent the batteries from over and under-charging we connect the energy source to the battery using a TriStar T-60 charge controller. We also use hardware sensors to measure the current flowing in and out of the battery and the battery voltage. Our experiments use the battery as a short-term buffer of five minutes.

Client Emulator. To experiment with a wide range of video traffic, we wrote a mobile client emulator in Java, which replays YouTube traces. For each video request in the trace file, the emulator creates a new thread at the specified time to play the video as per the specified duration. In addition, the emulator also generates synthetic video requests based on various configurable settings, such as available bandwidth, popularity distribution of videos, e.g., a Zipf parameter, viewing length distribution, and recommended list distribution.

Power Signal. We program our power supplies to replay solar and wind traces from our field deployment of solar panels and wind turbines. We also experiment with both multiple steady and oscillating power levels as a percentage, where 0% oscillation holds power steady throughout the experiment and N% oscillation varies power between (45 + 0.45N)% and (45 – 0.45N)% every five minutes. We combine traces from our solar/wind deployment, and set a minimum power level equal to the power necessary to operate GreenCache's always-active components (46W). We compress our renewable power signal to execute three days in three hours, and scale the average power to 50% of the cluster's maximum power.

6. EXPERIMENTAL EVALUATION

We first benchmark GreenCache's proxy and chunking overhead for our prototype. We then evaluate GreenCache's performance for real-world YouTube traces at multiple power levels with varying levels of oscillation. We then demonstrate the performance using realistic power traces from our energy harvesting deployment that have varying power and oscillation levels.

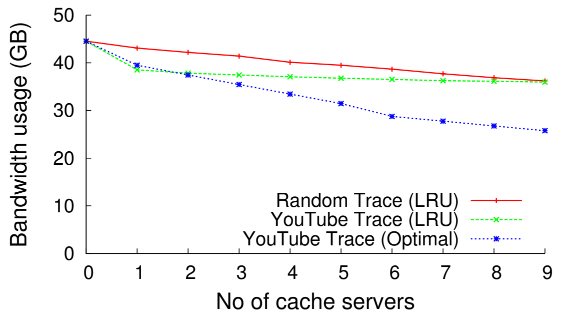
We use two metrics to measure the performance: (1) bandwidth usage between the cache and YouTube servers and (2) average buffering or pause time at the clients. Bandwidth usage denotes the total data received from backend servers over a given time interval; it also represents bandwidth cost that mobile operators must pay to Internet service providers. One primary objective of GreenCache is to reduce this bandwidth usage. Another key objective of GreenCache is to improve user's viewing experiences. Therefore, we consider average buffering time per video as our second metric to measure the performance. Note that our implementation tries to optimize both metrics independent of each other. However, note that optimizing for bandwidth usage does not depend on the power level, but on the total cache size, while optimizing for buffering time depends on both the cache size and the power level.

6.1 Benchmarks

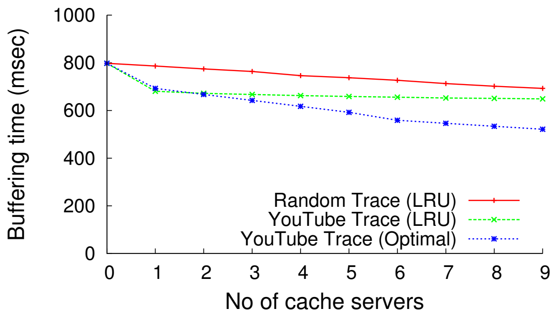
To measure the proxy's overhead, our client emulator creates a single thread and sends multiple video requests in succession. The breakdown of the latency overhead at each component for a sample 1 MB video chunk of 1 minute play length, assuming a 135 Kbps bit rate, is 30 ms at the proxy, 20 ms at the cache server, 50 ms in the network between the proxy and cache server, and 100 ms in the network between the proxy and client. The result demonstrates that the proxy's latency overhead is low. We also benchmark average buffering time for different blink intervals at various power levels. Table 1 shows the standard deviation, 90th percentile, and average buffering time for video requests, as the blink interval and power levels change. As expected, the buffering time increases with the blink interval at low to moderate power levels. We also benchmark the standard deviation, 90th percentile, and average buffering time for requests going to YouTube servers, which are as 150ms, 570ms, and 620ms, respectively.

To study the performance of our prototype cache for different cache sizes and power levels we take a 3 hour trace (from 7 PM to 10 PM on February 7th, 2012) from our 3 day YouTube trace. The trace contains a total of 8815 requests, for 6952 unique videos, over the 3 hour interval. Our trace reports the URL, video ID, client IP address, and request time for each video. In addition, we pull the recommended list for each video in the trace from the YouTube servers. Based on the video ID, its recommended list, client IP address, and the next requested video ID, we calculate the viewing length for each video. We assume the average video length as 5 minutes and the streaming rate as 135 Kbps. Also, we fix the downlink bandwidth from backend YouTube servers to the WiMAX station to 1 Mbps, and the storage capacity of each cache server as 1 GB. Further, we fix the blink interval as 60 seconds. We use a weighing factor of 0.6 for the proposed popularity-aware eviction policy.

First, we study the performance—bandwidth usage and buffering or pause time for clients—for different number of cache servers at full power for the real world 3 hour YouTube trace, as well as a synthetic trace of 8815 requests where each request is for a randomly chosen video from the aforementioned 6952 unique videos. In addition, we choose least-recently-used (LRU) cache eviction policy for this experiment; further, videos are not chunked. Figure 9 plots the total bandwidth usage and average buffering time for both random and real traces. We also plot the optimal performance for real traces assuming we know all requests in advance. The optimal policy always keeps most popular videos in the cache, and never evicts a popular video to store a less popular video (over a given interval).

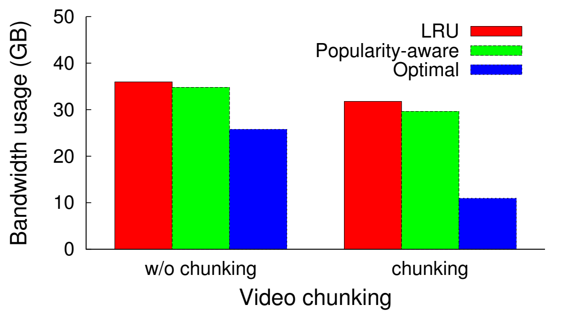


(a) Bandwidth usage

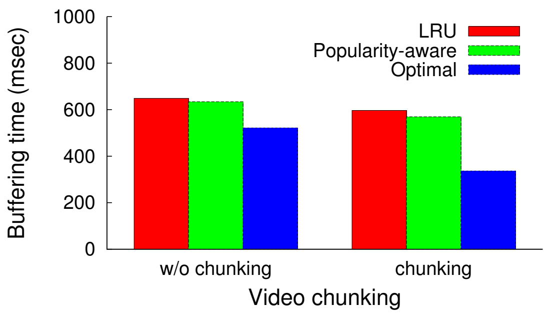


(b) Buffering time

Figure 9: Both bandwidth usage and buffering time reduce with increasing cache size.



(a) Bandwidth usage



(b) Buffering time

Figure 10: Video chunking reduces both bandwidth usage and buffering time.

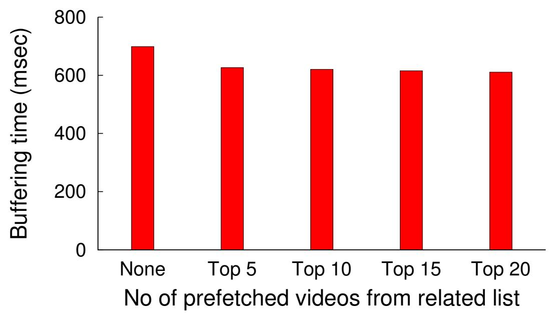


Figure 12: Buffering time decreases as the number of prefetched videos (first chunk only) from related lists increases.

As expected, the total bandwidth usage and average buffering time over the 3 hour interval decreases as the size or number of servers increases.

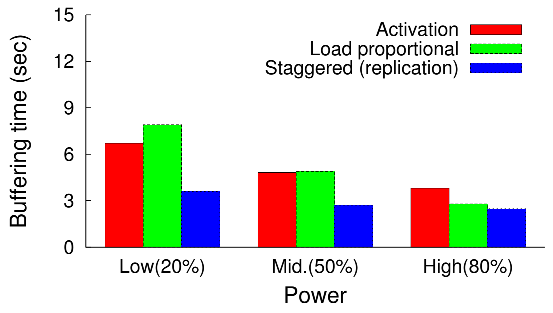
Next, to study the benefits of video chunking we measure the performance of three different cache eviction policies—LRU, popularity-aware, and optimal—for the 3 hour real trace at full power and 9 cache servers. Figure 10 shows that the performance of GreenCache's popularity-aware eviction policy is better ($\sim 7\%$) than that of LRU. Further, video chunking improves ($> 15\%$) the performance of all policies as it avoids storing unpopular chunks of popular videos. In all cases, LRU performs worse than others, which motivates our use of a popularity-aware cache eviction policy and video chunking for all further experiments.

6.2 Staggered load-proportional blinking

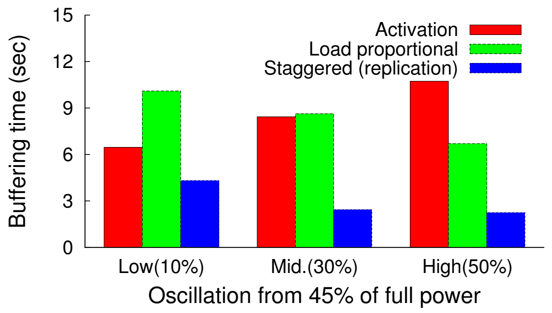
As discussed earlier, the total bandwidth usage over a fixed interval, as long as a request does not go to backend servers for an already cached video, does not depend on the available power level or blinking and layout policies; it only depends on the cache size and eviction policies. However, buffering time and users' experiences do depend on the available power, blinking and layout policies. In this section, we study the effects of the power level on the average buffering time, and various optimizations designed to reduce the buffering time. We use the same 3 hour real YouTube trace, as discussed above, and 9 cache servers for all further experiments. Further, we use video chunking and the popularity-aware eviction policy for all experiments.

To compare the proposed staggered load-proportional policy with the activation and load-proportional policies from Blink [11], we also implement an activation policy and a load-proportional policy for GreenCache, and integrate them with GreenCache's popularity-aware eviction policy, video chunking, and popularity-aware migration policy. The activation policy activates or deactivates servers as power varies, whereas the load-proportional policy distributes the power to servers in proportion to their popularity. Similar to the load-proportional policy, the activation policy also migrates popular chunks to active servers while deactivating servers due to the drop in the power level. Unlike the proposed staggered load-proportional policy, the load-proportional policy from Blink [11] does not replicate video chunks because it does not benefit from replication as it activates all servers at the same time every blink interval.

Figure 11(a) shows the average buffering time at different steady power levels. As expected, the activation policy performs better than the load-proportional policy at low power levels since, unlike the load-proportional policy, the activation policy does not incur the blinking overhead, which becomes significant in comparison to the active interval at low power levels. However, at moderate to high steady power levels, the benefit of a larger cache size, albeit blinking, dominates the blinking overhead for real-world traces. Furthermore, the buffering time decreases significantly if first chunks are replicated on all servers. Even at low power levels, replication of initial chunks significantly reduces the buffering time, while still leveraging the benefits of a larger cache size. Moreover, the perfor-



(a) Steady power



(b) Oscillating power

Figure 11: Buffering time at various steady and oscillating power levels.

mance of the staggered load-proportional policy remains almost the same at all power levels. As video popularity changes infrequently, migration overheads in our experiments are modest ($\sim 2\%$).

Figure 11(b) compares the average buffering time for the above policies at different oscillating power levels. We oscillate available power every five minutes. Since migration overhead of the staggered proportional policy is independent of power level, its performance remains almost the same at all oscillation levels. However, the activation policy incurs migration overhead whenever the number of active servers decreases. Consequently, the activation policy performs poorly at high oscillation levels, as indicated in the figure. Though replication of initial chunks reduces the buffering time at all power levels, it is primarily required at low power levels.

Next, we evaluate the benefits of prefetching initial chunks of related videos. As Figure 12 indicates, prefetching initial chunks of the top five videos reduces the buffering time by 10% as compared to no prefetching. Further, since prefetching more videos doesn't improve the buffering time, we limit the cache to prefetching only first chunks of top few videos from the related list. We choose to prefetch top five videos only in order to strike a balance between the performance gain and prefetching overhead.

6.3 Case study

To experiment with our WiMAX base station using a real WiMAX client, we use a Linux Desktop with Intel Atom CPU N270 processor and 1 GB RAM connected to Teltonika USB WiMAX Modem. We disable all network interfaces except the WiMAX interface. The desktop connects to the WiMAX base station (NEC Rel.1 802.16eBS), which we configure to route all video requests from the desktop to the proxy. We replay the same 3 hour YouTube trace on the WiMAX client, but we use real power traces from our solar/wind deployment, as described in the previous section, to power the GreenCache cluster.

Figure 13 plots average buffering time, calculated every five minutes, for three blinking policies: activation, load proportional, and staggered load proportional with first chunks replicated. As expected, the performance of all three policies goes down (buffering time goes up) when the available power drops down, and vice versa. However, the performance of activation degrades more than that of load-proportional when the available power drops down, since the activation policy incurs migration overhead when the number of active servers decreases. Further, replicating first chunks significantly reduces the buffering time for the staggered load-proportional policy at all power levels. Since the migration overhead of the staggered load-proportional policy is independent of power levels, its performance does not vary much, not even when the available power changes significantly, if first chunks are replicated.

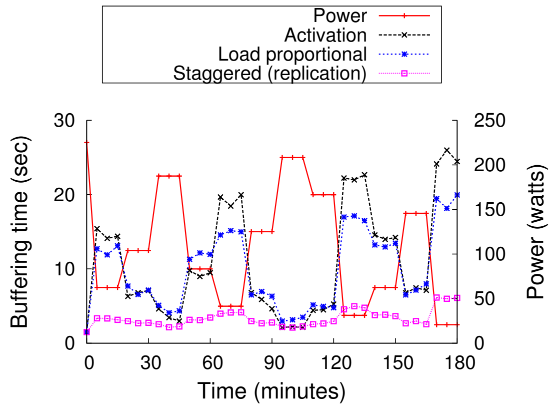


Figure 13: Buffering time at various power levels for our combined solar/wind power trace.

7. RELATED WORK

Blink [11] enables clusters to regulate their energy footprint to match available power without deactivating servers for long periods of time or incurring costly data migrations. Further, blinking policies provide several avenues to coordinate blinking among servers to maximize applications' performance at different power levels. In our previous work [11], we have presented a simple proof-of-concept example to show how a real-world, albeit simple and stateless, application could leverage blinking to perform well on intermittent power. Whether or not blinking is applicable to more complex, e.g., distributed and stateful applications, requiring real time performance guarantees remains an open research problem. In this paper, we propose a staggered load-proportional blinking policy with replication to maximize the performance of a distributed multimedia cache cluster running on intermittent power. We also use several video and traffic characteristics to design multimedia-specific optimization techniques for intermittent power.

As shown in Figure 10, knowing video requests in advance significantly improves the performance. Similarly, a better and more accurate energy-harvesting prediction technique would allow better cache management and could further improve the performance. For example, if we know that the available power is going to drop significantly in the next hour, we could perform costly popularity aware migration and replication before the power drops. In our previous work [12], we have proposed an energy-harvesting prediction technique that uses weather forecast to predict future energy, and shown that the forecast-based prediction performs better than existing prediction techniques based on past observed data. We plan to use our energy-harvesting prediction technique in combination with a network traffic forecast technique to further improve the performance of GreenCache as part of future work.

The use of caches to improve the performance of multimedia distribution systems has been studied extensively in the past two decades. [13] gives a general overview on existing multimedia caching techniques. Due to the vast amount of exiting work in this area, we only focus on the work closely related to our approach, although, to the best of our knowledge, there is no existing work that directly addresses multimedia caches for intermittent power.

Wu et al. [15] were among the first to propose the caching of chunks (segments) of a video. In contrast to our approach chunks are not equal in size and increase exponentially with the distance from the start of the video. The intention of this approach is to combine the number of consecutive chunks that are cached with the popularity of the video. E.g., for a very popular video all chunks would be stored on the cache while for less popular chunks only a certain number of the initial chunks of the video would be cached. Letting the chunk size grow exponentially has the advantage that the initial chunks of many videos can be stored without occupying too much of the caches storage space. Having only one or several initial chunks of a video stored on the cache bears the advantage that a requested video can be streamed to the client and played out without significant delay. Missing chunks can be streamed from the server immediately after the initial client request to allow for a smooth play out. In contrast to the approach presented by Wu et al., we decided for a scheme that splits all videos in equal sized chunks (except for the very last chunk) where the complete chunk can be transmitted to the client in a period that is equal or smaller than the blink interval, assuming a minimum transmission rate.

A more restrictive version of the caching of video chunks is the caching of the first chunk (prefix) only, which was introduced by Sen et al. [10]. The sole goal of this approach is to reduce the buffer time at the client, since the first chunk can be streamed from the cache much faster than from a remote server. Our initial work on prefix prefetching of videos listed on YouTube's related video list [8] is based on this approach, but proactively prefetches prefixes instead of caching them. As we have shown in [8], prefix prefetching can significantly improve the viewer's experience of watching videos and this motivated us to investigate how the prefetching approach performs on a multimedia cache for intermittent power. The results presented above show that prefix prefetching can improve the experience of a viewer also in the case of a blinking multimedia cache.

As in our current work, trace-based driven simulations are also used in [5] and [17] to investigate the effectiveness of caching for YouTube videos. Both investigations show that caching of YouTube video can both, on a global and regional level, reduce server and network load significantly. In contrast to the work presented in this paper, both studies do not consider scenarios in which power for the caches is intermittent.

8. CONCLUSION

This paper presents techniques for optimizing multimedia caches running off intermittent renewable energy sources. These caches are important in improving the performance of "off the grid" cellular towers and base stations, which are increasingly common in developing countries that have both a lack of infrastructure and an unstable grid. We show how combining a blinking abstraction, which rapidly transitions cache servers between an active and inactive state, with techniques for chunking (or segmenting) videos, replicating popular chunks, and staggering server active intervals improves the performance of these caches when compared to standard techniques for powering servers *on* and *off* based on available power. Our work demonstrates that running multimedia caches off intermittent power from renewables poses interesting new research problems. Our results show that GreenCache's blinking techniques decrease both backhaul bandwidth and client access latency compared to existing approaches that simply activate and deactivate servers. In the latter case, resulting in 3X less buffering (or pause) time by the client watching a video.

9. REFERENCES

[1] Endace DAG Network Monitoring Interface. http://www.endace.com/.

[2] Stoke solutions: Mobile data offload. http://www.stoke.com/news/pr/2011/pr020911.asp.

[3] Youtube press statistics. http://www.youtube.com/t/press_statistics, September 2012.

[4] W. Balshe. Power System Considerations for Cell Tower Applications. http://www.cumminspower.com/www/literature/technicalpapers/PT-9019-Cell-Tower-Applications-en.pdf, 2011.

[5] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *IMC*, October 2007.

[6] J. Erman, A. Gerber, K. K. Ramadrishnan, S. Sen, and O. Spatscheck. Over the Top Video: The Gorilla in Cellular Networks. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 127–136, New York, NY, USA, 2011. ACM.

[7] J. Guay. India: Forget the Grid, Community Power is Here. http://ourworld.unu.edu/en/india-forget-the-grid-community-power-is-here/, August 2012.

[8] S. Khemmarart, R. Zhou, L. Gao, and M. Zink. Watching User Generated Videos with Prefetching. In *MMSys*, February 2011.

[9] G. P. Office. Open Virtualized WiMAX Base Station Node for Wide-Area Wirelesss Deployments. http://groups.geni.net/geni/wiki/WiMAX, June 2011.

[10] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caching for xiMultimedia Streams. In *INFOCOM*, 1999.

[11] N. Sharma, S. Barker, D. Irwin, and P. Shenoy. Blink: Managing Server Clusters on Intermittent Power. In *ASPLOS*, March 2011.

[12] N. Sharma, J. Gummeson, D. Irwin, and P. Shenoy. Cloudy Computing: Leveraging Weather Forecasts in Energy Harvesting Sensor Systems. In *SECON*, June 2010.

[13] X. Tang, J. Xu, and S. Chanson. *Web Content Delivery*. Springer, 2005.

[14] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and Z. Zhu. Delivering Energy Proportionality with Non Energy-Proportional Systems – Optimizing the Ensemble. In *HotPower*, December 2008.

[15] K.-L. Wu, P. S. Yu, and J. L. Wolf. Segment-based Proxy Caching of Multimedia Streams. In *WWW*, 2001.

[16] Q. Xu, J. Huang, Z. Wang, F. Qian, A. Gerber, and Z. Mao. Cellular Data Network Infrastructure Characterization and Implication on Mobile Content Placement. In *SIGMETRICS*, June 2011.

[17] M. Zink, K. Suh, Yu, and J. Kurose. Characteristics of Youtube Network Traffic at a Campus Network - Measurements, Models, and Implications. *Elsevier Computer Networks*, 2009.