Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices

Fahad R. Dogar
Carnegie Mellon University
Pittsburgh, PA, US
fdogar@cs.cmu.edu

Peter Steenkiste
Carnegie Mellon University
Pittsburgh, PA, US
prs@cs.cmu.edu

Konstantina Papagiannaki
Intel Labs, Pittsburgh, PA, US
dina.papagiannaki@intel.com

ABSTRACT

Energy management is a critical issue for mobile devices, with network activity often consuming a significant portion of the total system energy. In this paper, we propose Catnap, a system that reduces energy consumption of mobile devices by allowing them to sleep *during* data transfers. Catnap exploits high bandwidth wireless interfaces – which offer significantly higher bandwidth compared to available bandwidth across the Internet – by combining small gaps between packets into meaningful sleep intervals, thereby allowing the NIC as well as the device to doze off. Catnap targets data oriented applications, such as web and file transfers, which can afford delay of individual packets as long as the overall transfer times do not increase. Our evaluation shows that for small transfers (128kB to 5MB), Catnap allows the NIC to sleep for up to 70% of the total transfer time and for larger transfers, it allows the whole device to sleep for a significant fraction of the total transfer time. This results in battery life improvement of up to 2-5x for real devices like Nokia N810 and Thinkpad T60.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Measurement, Performance, Experimentation

Keywords

Energy Efficiency, System Power Management, Traffic Shaping, Wireless, Mobile Devices

1. INTRODUCTION

Energy efficiency has always been a critical goal for mobile devices as improved battery lifetime can enhance user experience and productivity [25, 23]. A well-known strategy for saving power is to sleep during idle times. Prior work in this context exploits idleness at various levels, such as sleeping during user think time [8] or when TCP is in slow start [20]. Recent work also explores the possibility of entering low power consuming states in the middle of data transfers [21]. However, deeper sleep modes, such as 802.11 PSM or Suspend-to-RAM (S3), are assumed to be unusable *during* data transfers when applications are constantly sending data, as entering into these sleep states degrades application performance due to the overhead of using these sleep modes [5, 20].

In this paper, we propose Catnap, a system that saves energy by combining tiny gaps between packets into meaningful sleep intervals, allowing mobile clients to sleep *during* data transfers. These tiny gaps arise when a high bandwidth access link is bottlenecked by some slow link on the path to the Internet. A typical example is a home wireless network where 802.11 offers high data rates (54Mbps for 802.11g and 300Mbps for 802.11n) but the uplink to the Internet (Cable, DSL) offers much lower bandwidth, typically in the range of 1-6 Mbps. Catnap is targeted towards data oriented applications (e.g., web, ftp, etc) that consume data in *blocks*, so combining small idle periods and hence delaying individual packets is acceptable if the data block is delivered on time. These blocks are often called *Application Data Units* (ADU) [10].

Catnap builds on three concepts. First, it *decouples* the wired segment from the wireless segment, thereby allowing the wireless segment to remain inactive even when the wired segment is actively transferring data [14, 6]. We call the functional component separating the wireless and wired segment a "middlebox", a capability that could easily be integrated with a home wireless Access Point (AP)[1]. The middlebox will batch packets when the mobile device is sleeping and send them in a burst when the device wakes up. Second, it uses an ADU (e.g., web object, telnet command, P2P chunk, etc.) as the unit of transfer. This allows the middlebox to remain *application independent* and yet be aware of *application requirements*, thereby ensuring that delay sensitive packets (e.g., telnet) are differentiated from delay tolerant packets (e.g., web, ftp). Third, the middlebox uses bandwidth estimation techniques [18] to calculate the available bandwidth on the wired and wireless links. The bandwidth estimates determine the amount of batching required for on-time ADU delivery. These concepts have been used in dif-

[1]Note that such capability does not necessitate the introduction of a new device, but could easily be supported by other existing devices on the path between the wireless client and its intended wired destination.

Application Type	Examples	Catnap Strategy (Target Device)	Expected Benefits	Remarks
Interactive	VoIP	None	None	Device and NIC remain up to maintain user experience
Short Web Transfers (<100kB)	`facebook` `google`	Batch Mode (Small Devices: Smart Phones, Tablets, etc.)	30% NIC sleep time (Section 6)	- Embedded objects rendered together - Batching may require some app. specific support at the proxy (e.g., java scripts)
Medium Sized Transfers (128kB to 5MB)	`you-tube` `flickr` images mp3 songs	Normal Mode (Small Devices)	up to 70% NIC sleep time (Section 5.4)	No impact on user experience
Large Transfers (>5MB)	maps & movies software updates ISR [19]	1. Normal Mode (Small Devices) 2. S3 Mode (All Devices)	1. >70% NIC sleep time 2. 40% device sleep time for a 10MB transfer (Section 5.4)	1. No impact on user experience 2. User permission to enter S3 mode is desirable (Section 7)

Table 1: Energy Savings with Catnap for different types of applications. Evaluation numbers correspond to an 802.11g wireless access network with a 1.5Mbps wired Internet connection. NIC sleep mode targets smaller devices, such as tablets and smart-phones, where the NIC activity can consume up to 60% of the total energy [25]. S3 mode is applicable for all mobile devices (including laptops) as the whole device goes into sleep state, consuming negligible power.

ferent contexts but are combined in Catnap in a novel way to create sleep opportunities.

We have implemented a prototype of Catnap as an application independent process at the middlebox. This process resembles the functionality of a proxy in acting as a "translational" functional component, but supports several other functions as we explain later. For lack of a better term we will call it a "proxy" in what follows.

The proxy uses traditional transport layer connection splitting to decouple wired and wireless segments, but supplements OS buffers with extra storage capacity. This ensures that data continues to flow on the wired segment even if the client is in sleep mode for arbitrary long intervals of time. Catnap further implements a novel *scheduler* that schedules ADU transmissions on the wireless segment, ensuring maximum sleep time for the clients with little or no impact on the end-to-end transfer time. The scheduler dynamically *reschedules* transfers if conditions change on the wired or wireless segments, or new requests arrive at the proxy. The scheduler can further operate in two modes: i) one where no increase in end-to-end transfer completion time is allowed, and ii) a second one, that we call *batch* mode. In this latter case, multiple small objects are batched together *at the proxy* to form one large object, allowing additional energy savings at the cost of increased delay for individual objects.

1.1 Energy Savings with Catnap

Table 1 gives an overview of Catnap's benefits for various applications, also highlighting some of the possible issues with using Catnap. As we can see from the table, for transfer sizes larger than 128kB, Catnap can put the NIC to sleep for almost 70% of the total transfer time without any impact on user experience. For transfers larger than 10MB, S3 mode provides significant system wide energy savings. However, its use is limited to scenarios where the user is not involved in other tasks. Our evaluation further demonstrates the ability of Catnap to benefit existing application protocols, such as HTTP and IMAP, without requiring modifications to servers or clients. Performance and energy efficiency are sometimes at odds in these application scenarios that often involve transfer of very small blocks of data (e.g., short web transfers). We demonstrate using the Catnap *batch* mode that one can manage such a trade-off for very small transfers by batching multiple objects at the proxy.

1.2 Contributions

The key **contributions** of this paper are:

- We design and implement Catnap which exploits the bandwidth discrepancy between wireless and wired links to improve energy consumption of mobile devices. The key design components of Catnap include an application independent proxy that decouples wired and wireless segments, and a scheduler that operates on ADUs and attempts to maximize client sleep time without increasing overall ADU transfer times.
- We conduct a detailed evaluation of Catnap using real hardware, under realistic network conditions. We show how Catnap can improve battery life of tablet PCs and laptops by enabling various sleep modes. We also consider case studies of how existing application protocols, like HTTP and IMAP, can use and benefit from Catnap.

Our results clearly attest to the promise of Catnap to increase mobile client battery lifetime at no to little performance impact.

1.3 Paper Organization

The rest of the paper is organized as follows. In Section 2, we motivate the need for Catnap. We present the design and implementation of Catnap in Sections 3 and 4. Section 5 presents the detailed evaluation of Catnap. In Section 6, we present case studies that show how two legacy application protocols (IMAP and HTTP) can use and benefit from Catnap. We discuss the usability aspects of Catnap in Section 7. Finally, we discuss some future extensions to Catnap and present our conclusions.

2. MOTIVATION

We first motivate the need for a system wide view of energy management with a focus on why existing sleep modes are often not very useful in practice. We then discuss an opportunity for energy savings and how we can exploit it to

Suspend-to-RAM (S3)	802.11 PSM
10 sec	Beacon Interval (≈ 100ms)

Table 2: Overhead of using different sleep modes.

enable various sleep modes without degrading application performance.

2.1 Sleep Modes – Potential Benefits and Limitations

Many devices today offer a range of sleep modes that allow sub-components of the system to be placed in various sleep states. The benefits of putting specific sub-systems into sleep mode depend on the contribution of each subsystem to the overall energy consumption. For example, 802.11 consumes less than 15% of the total power on a laptop but can consume more than 60% power of the total power on a small hand-held device [25]. As a result, the benefits of 802.11 power saving mode (PSM) depend heavily on the device in use. On the other extreme are deep sleep modes like S3 (Suspend-to-RAM) that benefit both small as well as large devices, since they shut-off most of the device circuitry, thereby hardly consuming any power at all. There are also other sleep modes that are becominge increasingly more popular, such as a "reading mode" that keeps the display on but puts most of the other sub-systems into sleep mode, or a "music mode" that can be used if the user is only listening to music and not doing any other task. As network activity is often an important part of the user's interaction with the device, managing it in an intelligent manner can be critical for increasing the battery life of mobile devices.

Limitations of Power Saving Modes: In today's Internet, it is difficult to utilize the available sleep modes *during* data transfers because of their associated overheads. Table 2 lists the overhead of entering and leaving the sleep state when using Suspend-to-RAM(S3) and 802.11 PSM modes. For the S3 mode, the associated overhead approaches 10 seconds [4], and is clearly unsuitable for TCP based applications. Similarly, for 802.11 PSM, the NIC wakes-up at the granularity of beacon intervals (100ms); as packets can be delayed by this interval, performance of applications like telnet, VoIP, distributed file access, or short web transfers can be severely degraded if the NIC goes into sleep mode when these applications are in use [5, 20]. Therefore, there is general consensus that the NIC *should not* enter PSM mode as soon as it encounters an idle period – rather it should only enter sleep mode when this will not impact application performance [5, 20, 21].

Making the PSM implementation aware of application latency requirements is problematic for various reasons. First, it hinders application innovation as policies for new applications have to be incorporated in the PSM. Second, a single application, like web, could support streaming, instant messaging, and file transfers, that have very different interactivity requirements. Therefore, most 802.11 cards use a conservative policy of entering PSM only when there is no network activity for a certain time (generally on the order of 100ms). This ensures that cards do not sleep in the middle of an application session, even for applications that do not send packets continuously (e.g., VoIP).

We conducted an experiment to verify this for a 802.11g based network with a cable modem providing the uplink connection. As shown in Table 3, applications like ping and ssh

	ping	ssh	web-transfer	ftp
Total Time (sec)	10	20	5	100
Sleep Time (sec)	7	12	0	0

Table 3: Transfer and NIC sleep times for different applications in a typical home wireless scenario. Applications that constantly send data (web and file transfer) do not allow any NIC sleep time.

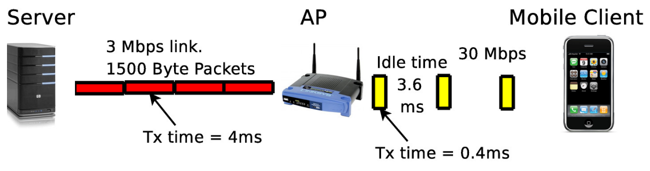


Figure 1: Data transfer in a typical home wireless scenario. Idle periods on the wireless segment are individually too short and remain un-utilized.

that have idle periods, either because of inherent 1sec gaps between successive packets or due to user think time, can indeed allow the NIC to sleep throughout their execution. On the other hand, data intensive applications, like file or web transfer, tend to send data as fast as possible and thus will hardly lead to any significant idle periods that could allow the NIC to enter sleep mode. As these data intensive applications become increasingly more popular, we need to find avenues where we can save power even while using these applications.

2.2 Opportunity – High Bandwidth Wireless Interfaces

Figure 1 depicts one such opportunity that can allow energy savings *during* the use of data oriented applications. It shows a typical 802.11g based home wireless network that connects to the Internet through a DSL connection. The server is constantly sending packets to the access point (AP) but packets sent on the wireless channel are roughly spaced by the packet transmission time on the bottleneck link (DSL). For the example scenario, this idle time is less than 4 millisecond. As discussed and shown earlier, 802.11 PSM does not exploit these small idle opportunities and the NIC remains up for the whole transfer duration. Of course, using S3 mode in this scenario is impossible as TCP will likely time-out during the time the client is in S3 state.

Discussion: It is difficult to exploit the above opportunity in today's Internet. The key reasons are as follows:

- It is difficult for the network/middlebox to differentiate between packets that are needed immediately by the client from packets that can be potentially delayed. For example, a telnet packet should be delivered immediately while a packet that is part of an HTTP response message can be delayed until the whole message is received. This implies that applications should be able to provide a *workload hint* to the network/middlebox indicating when the data is consumed by the application.

- The current TCP based communication model assumes strict synchronization between end-hosts. Not only the two end-points need to be present at all times but they

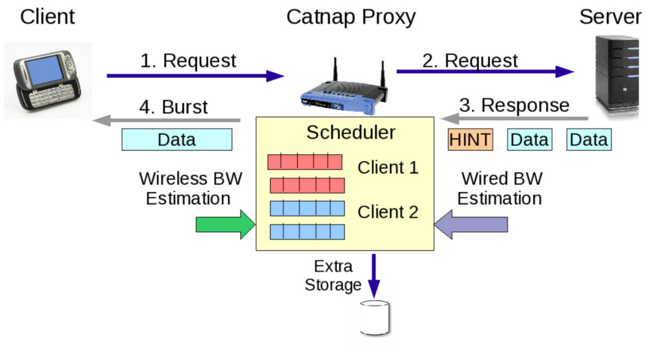


Figure 2: Basic overview of Catnap.

also need to send regular messages (e.g., ACKs) in order to sustain communication. This limits the opportunities to sleep for a mobile client, as going into sleep mode could result in degradation in performance or even loss of connection. The system should be able to support *relaxed syncronization* between end-points, allowing middleboxes to *shape traffic* in a way that allows maximum sleep time for the mobile device.

3. CATNAP: DESIGN

Catnap addresses the above two problems, thereby exploiting the opportunity created by bandwidth discrepancy between the wired and wireless segments. We first provide a high level overview of Catnap followed by details on its main components.

3.1 Overview

Figure 2 provides the high level overview of Catnap by showing the key steps involved in a typical data transfer. The client application establishes a transport connection with the Catnap proxy. It then sends the request to the proxy that checks whether the request can be served through the local cache. If this is not the case, the proxy forwards the request to the destination server over a separate transport connection (proxy - server). The server sends the response to the Catnap proxy that schedules the best time to initiate the wireless transmission to the client, so as to minimize the time the client is awake.

The timing of the transmission is based on what we term as *workload hints*. *Workload hints* expose information on the ADU boundaries in the provided server response and the willingness of the client to suffer a slight increase in overall transfer completion time for additional energy savings. The former can be provided by the server itself or through an application specific proxy on the middlebox. We go through the *workload hints* in more detail later in this section.

The scheduler uses the *workload hints* along with information about the wired and wireless bandwidth to calculate the total transfer time and the length of the time slot needed to transmit the block of data on the wireless side. In the general case, the wireless time slot is scheduled as late as possible while ensuring that transfer times do not increase. Of course, the network conditions can change or new requests overlapping with the previous time slot can arrive, so the scheduler may need to update its scheduling decision as new information becomes available. The Catnap scheduler can further operate in what we call the *batch* mode. If the client is tolerant to a slight increase in the transfer completion time, then the scheduler can concatenate multiple small slots into bigger ones, thus allowing for greater energy savings. The *workload hint* will convey the willingness of the client to employ the *batch* mode.

Finally, while this paper focuses on Catnap's benefits in download scenarios, the case of uploading data from the mobile device to the Internet is even simpler and does not require the full Catnap functionality. In an upload scenario, the device bursts the data to the proxy that buffers it and sends it at the bottleneck rate to the destination. The Catnap scheduler does not need to be involved in this case.

3.2 Example Scenarios

We discuss two example scenarios in which Catnap can provide energy savings: i) small transfers where the NIC can be put into sleep mode, ii) larg downloads where the whole device can be put into sleep using the S3 mode. In both examples, we quantify the potential energy savings under the network configuration shown in Figure 1.

NIC energy savings for small transfers ($<$ 1MB): Examples of such small transfers include many web-pages, high resolution images, and initial buffering requirements of video streaming (e.g., `you-tube`). For a 128kB transfer, the total time required to complete the transfer is 350ms while it takes only 35ms to complete the transfer on the wireless link. This means that the remaining time (315ms) can be used by the mobile device to sleep. As we show in Section 5, this time is long enough for the wireless NIC to enter sleep mode and then return without any change to the PSM implementation. As a result, the user does not experience any change in response times.

S3 sleep mode for large transfers ($>$ 5 MB): Examples of larg transfers include: downloading a song or a video, downloading high resolution maps, file synchronization between devices, or downloading a virtual machine image to resume a suspended session (e.g., ISR [19]). If we assume a 10MB transfer, the total transfer time is 27 seconds. The wireless transmission takes only 2.7 seconds. The mobile device can thus sleep for the remaining time (24.3 seconds). A duration of tens of seconds is long enough to enter and exit S3 mode. We show in Section 5 how S3 mode can provide significant energy savings for large file transfers.

The important thing to note here is that we can only use S3 mode if the user is not doing any other task while the download is taking place. For example, the user starts the download and leaves the device to grab coffee, or the user stops at a gas station and starts downloading mp3 songs while she refuels her car. In such cases, as the user is not actively using the device, the system can easily use the S3 mode. In some scenarios, the user may be so short of battery that she is willing to stop other tasks in order to get energy savings from the S3 mode. We discuss the usability aspects of the S3 mode later in Section 7.

3.3 Proxy

Catnap achieves *decoupling* between the wired and wireless segments by introducing a proxy at the middlebox. The proxy uses a combination of split connection approaches [6] and DTN-style in-network storage [14]. This is important because in some cases we want the Catnap proxy to operate in a cut-through way i.e., to forward data as soon as it receives it, which is similar to the split connection approaches. However, in some cases the proxy may need to buffer a large amount of data before sending it to the client. This case is similar to the DTN-style store and forward approach that requires extra storage to supplement the OS buffers.

Supporting the above form of decoupling allows Catnap to leverage the well understood benefits of split connection approaches like independent congestion and error control on the two segments, and the high performance of cut through data transfers [6]. In addition, extra storage at the middlebox for buffering data also ensures that even if the wireless segment is inactive, data continues to flow on the wired segment at the maximum rate and is not flow controlled. This form of decoupling, however, may create problems due to the additional state at the middlebox. Specifically, it can introduce new middlebox failure modes that can break application semantics or make it difficult to handle mobility *across* different subnets [6]. Such problems already exist due to the widespread deployment of middleboxes (e.g., NATs) and can be addressed using enhancements to the Internet architecture [12].

Deployment Scenarios: As data oriented applications get more popular, the storage requirements of a Catnap proxy will increase. Many modern APs already support the addition of USB sticks that can provide this extra storage for Catnap [1]. In certain deployments (e.g., enterprise settings) that use thin APs and a centralized controller, we expect the Catnap functionality and the storage to be co-located with the centralized controller. This provides an easy way to deploy Catnap without modifying all the APs in an enterprise. The main point is that the Catnap proxy should be placed at a point that can decouple the end-to-end path into two segments: a bottleneck segment on the one side and a high bandwidth segment on the other side. So the proxy need not always be physically co-located with the wireless AP. This implementation strategy also naturally handles mobility *within* a subnet, as data is buffered at the central controller rather than at individual APs. However, the scheduling decision may need to be updated after a handoff, as the wireless bandwidth available to the new AP may significantly differ from the bandwidth available to the old AP.

3.4 Workload Hints

Catnap relies on the concept of ADU to identify when a certain data block will be used by an application [10]. The concept of ADU is well known in the networking community as it is a natural unit of data transfer for applications. Different applications use different ADUs – for example, P2P applications can define a *chunk* as an ADU because the download and upload decisions in P2P applications are made at the granularity of a chunk. Similarly, a file transfer application can define the whole file as an ADU because a partial file is generally useless for the users. In some contexts, defining an ADU may vary depending on the scenario. For example, a web server can define all objects embedded within a web-page as one ADU if all the objects are locally stored or as separate ADUs if these objects need to be fetched from different servers. ADUs also have other advantages – for example, they are a natural data unit for caching at the middlebox

While applications naturally operate on ADUs, data transfer is based on byte streams (TCP based sockets) rather than ADUs. In order for Catnap to estimate the oncoming workload, we need to identify the boundary between ADUs. Such information can be provided by the application or through an application specific proxy on the middlebox. Note that most applications have an implicit notion of ADUs embedded in their byte stream. For example, as we discuss in Section 6, both HTTP and IMAP implicitly define their responses as ADUs – they also have headers that include the length of the ADUs in them. Such information could be easily extracted by an application specific proxy on the middlebox.

In the case when *workload hints* are provided by the servers, each ADU is preceded by a short header. The header comprises three fields: `(ID, Length, Mode)`. The `Length` field is the most important as it allows the proxy to identify ADU boundaries, enabling the proxy to know when the ADU will be "consumed" by the application. The `Mode` field indicates the client's willingness to use *batch* mode. In our current design, knowledge on `Mode` is sent from the client to the proxy and played back by the server. The default mode is not to use batching. Finally, the `ID` field is optional and is used to identify and cache an ADU.

3.5 Scheduler

The scheduler is the heart of Catnap as it determines how much sleep time is possible for a given transfer. The goal is to maximize the sleep time without increasing the total transfer time of requests. This requires *precision* in scheduling and the ability to dynamically *reschedule* requests if the conditions change. The rescheduling of requests also ensures fairness amongst requests as old requests can be multiplexed with a new request that arrives later in future.[2]

As we do not want to increase the transfer time, we rule out the simple store and forward technique where the middlebox receives the whole ADU and *then* transfers it to the mobile client. As this strategy is agnostic of network conditions, it could add significant delay if the wireless medium is temporarily congested. In some cases, however, the client may be willing to tolerate additional delay if this improves the battery life. The user would, however, want control over this option, so it can be used in a judicious manner. Based on these requirements, the scheduler provides two modes: in the *normal* mode it allows maximum sleep time without increasing the transfer time, while in the *batch* mode it provides savings on top of the normal mode by batching (and thus delaying) data. We will first focus on the normal mode and then discuss how we provide additional support in the form of batching.

How much can we sleep? There are *four* variables that determine the amount of sleep time that is possible for a given transfer: i) the size of the transfer, ii) the cost (in terms of time) of using a sleep mode, iii) the available bandwidth on the wired segment, and iv) the available bandwidth on the wireless segment.

The first two variables are easier to determine. The hints provide the Catnap proxy with the size of the transfer. Similarly, the cost of entering a sleep mode is generally fixed for a certain sleep mode and device. The last two factors i.e., available bandwidth on the wired and wireless segments, are

[2]For fairness amongst different stations (different Catnap proxies or legacy 802.11 stations), we rely on the fairness provided by 802.11.

more difficult to determine. Given these four variables, the sleep time for a given transfer is given by:

$$T_{Sleep} = \frac{Size_{ADU}}{BW_{Wired}} - (\frac{Size_{ADU}}{BW_{Wireless}} + Cost_{SleepMode}) \quad (1)$$

The first term is the time required to complete the transfer on the wired segment. If the wired segment is the bottleneck then this time is also the total transfer time. We can only sleep if this time is greater than the sum of the transfer time on the wireless segment and the cost of using the sleep mode. The equation provides some intuition for Catnap. For example, sleep mode is only possible when the wireless bandwidth is *more* than the wired bandwidth. Similarly, because of the fixed cost of using a sleep mode we need a minimum size for the ADU to get benefits from sleeping. As we increase the ADU size this cost is amortized and we can get more time to sleep. The last observation provides the motivation behind Catnap's batch mode.

Challenges: Leveraging the above sleep time benefits involves several challenges. First, calculating the available bandwidth is difficult, especially for the wireless medium, which is a shared resource. Conditions on the server side can also change due to extra load that may impact the finish time of transfers. A second, but related, challenge is that transfer requests come in an *online* manner – the proxy does not have a-priori information about future requests. So Catnap's policy of waiting and batching data may prove suboptimal as new requests may arrive during the batching period. In summary, the Catnap scheduler has to account for unpredictability in network conditions as well as the online nature of the requests arriving at the proxy.

1. Estimate **Capacity** of Wired Link
2. Estimate **Available BW** for this transfer
3. Calculate **finish time (FT)** of transfer
4. Estimate **Wireless Capacity**
5. Calculate **Virtual Slot Time (VST)**
6. Schedule Transfer at (FT − VST)
Shift or **Merge** Slots, if required
7. Periodically check for **Rescheduling**

Figure 3: Basic steps of the scheduling algorithm.

3.5.1 *Algorithm*

Figure 3 lists the basic steps of Catnap's scheduling algorithm. It accounts for conditions on both the wired and wireless medium in deciding when to schedule a transfer. Furthermore, it reschedules requests based on changes in network conditions (both wired and wireless) or the arrival of new requests.

The two key features of the scheduling algorithm are: i) it mostly relies on information that is *already present* with the Catnap proxy to estimate the finish time of the transfer, as well as the slot time on the wireless segment and ii) it avoids *multiplexing* time slots for different clients on the wireless side, as this results in extending the slot time for all the clients that are multiplexed.[3]

[3]As an example, consider two clients with a transfer duration of 3ms each. Multiplexing them together means that both of them are up for approximately 6ms whereas without multiplexing they only need to be up for 3ms.

We now describe the scheduling steps in detail. Recall that the scheduling decision is made when the Catnap proxy receives the response that has the workload hint about the length of the transfer. With this information, the scheduler performs the following steps:

Step 1 – Estimate wired capacity. On the wired side, the bottleneck could be the access link, a core Internet link, or the server. Earlier studies have shown that broadband access links are often the bottleneck in the end-to-end path [11]. Therefore, as our main target is home wireless scenarios, it is reasonable to start-off with the assumption that the access link is the bottleneck in the wired path. We can use a fixed value if the access capacity remains constant (e.g., DSL) or we can periodically use an existing bandwidth estimation tool to estimate the capacity of the access link [17]. To account for the case that the bottleneck is elsewhere, e.g., on the server, the Catnap proxy continuously monitors the throughput that is actually achieved on the server-proxy path.[4] If the observed throughput differs from the initial bandwidth estimate, the transfer is re-scheduled (Step 7) using the observed throughput as the new estimate.

Step 2 – Estimate Available BW for the transfer. When estimating the available bandwidth for a transfer, we need to account for the fact that multiple transfers may be using the access link at the same time. This is necessary when we use the access link bandwidth as an initial estimate of available bandwidth. To this end, the proxy identifies all active transfers and splits the access bandwidth among these transfers in different proportions. As the transfers are TCP based, we divide the bandwidth in an RTT-proportional way amongst the ongoing transfers. Note that bandwidth estimates based on observed throughput automatically account for the sharing of the access link, so if this estimate is incorrect, it will be updated based on actual observed performance (Step 7).

Step 3 – Calculate transfer finish time. Recall that the scheduler already knows the length of the transfer (`Length`). Step 2 also provides information about the available bandwidth for the transfer on the wired side. So combining these two pieces of information allows the scheduler to determine the finish time (FT) of the transfer. This time is important since it constitutes the scheduler's deadline for the client transfer.

Step 4 – Estimate wireless capacity. Based on steps 1-3 we know how long the transfer will take on the wired server-proxy segment. We also need a similar estimate for the wireless segment and estimating the available wireless capacity is a first step in this direction. Note that the conditions may change, but we use the *current* available capacity as a guide to what we will likely experience in the future. As wireless is a shared resource and the proxy can potentially overhear other transmissions in its neighbourhood, our prototype uses the medium busy/idle time as the basis for wireless capacity estimation. We provide more details on our specific implementation in Section 4.3.

Step 5 – Calculate Virtual Slot time. Estimating the wireless capacity allows the proxy to know how much channel access it will get on the wireless medium (Step 4). We then combine this information with the bit-rate used between the client and the AP to estimate the expected throughput between the proxy and the client. Finally, based

[4]We ignore this step for short transfers due to the effects of TCP slow start.

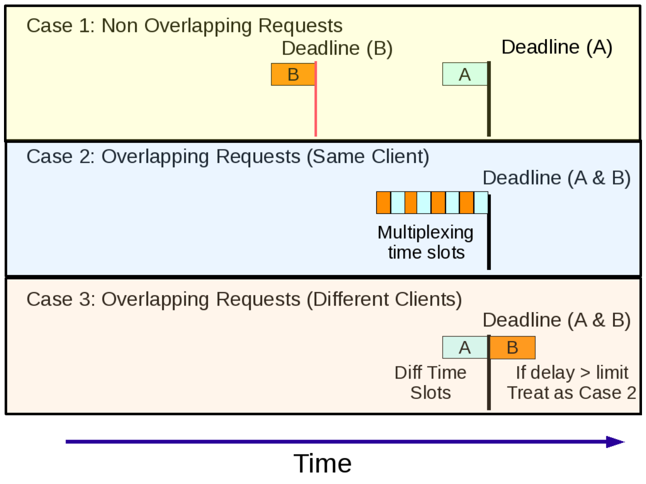

Case 1: Non Overlapping Requests
Deadline (B)
Deadline (A)
B
A
Case 2: Overlapping Requests (Same Client)
Deadline (A & B)
Multiplexing time slots
Case 3: Overlapping Requests (Different Clients)
Deadline (A & B)
A
B
Diff Time Slots
If delay > limit Treat as Case 2
Time

Figure 4: Different cases of virtual time slot scheduling.

on the expected throughput and the size of the transfer, we know the time required to burst the whole transfer on the wireless segment. We call this the virtual slot time (VST). Note that we also add a small padding time to each virtual time slot as a safety measure to account for unexpected changes to the network conditions.

Step 6 – Schedule the Virtual Slot. Ideally, we want to schedule the virtual time slot as late as possible so that its end time aligns exactly with the finish time of the request. If there is no other transfer scheduled at that time then we can simply schedule the virtual time slot at this position (Figure 4: Case 1).

However, there will be cases when multiple transfers contend for wireless transmission time, e.g., a new transfer overlaps with a previously scheduled transfer. In such cases, if the new and old slots belong to the *same* client then we simply multiplex them together and make a bigger time slot (Case 2). The case where the slots partially overlap is also handled naturally in this case.

In contrast, if the two slots belong to different clients, then multiplexing them together is not the best strategy as it increases the up time for *both* clients. So if the time slots correspond to different clients then we should schedule them at non-overlapping times. However, non-overlapping time slots mean that at least one of the time slots is scheduled *after* its original finish time (Case 3). We can pick any order for scheduling the requests (A-B or B-A), if the slots perfectly overlap. If the slots partially overlap then we first schedule the transmission that has an earlier deadline.

We adopt this strategy of non-overlapping time slots if the increase in delay is less than a certain threshold, otherwise we multiplex the two time slots. This threshold can be tuned to achieve greater energy savings at the cost of some increase in delay.[5] For example, a very small value of the threshold can be used if the user is not willing to tolerate any additional delay and a higher value of the threshold may be fine if the user is running short of battery. We discuss possible ways to capture such user preferences in Section 7.

A final point is that multiplexing two slots increases the slot length and the new, bigger time slot may start overlapping with some other slot, so this decision needs to be made recursively. In practice, implementations can put a limit to the number of slots that can be *disturbed*. If this limit is exceeded, then a slot is scheduled even if the scheduled completion time exceeds the finish time of the request.

[5]Our current implementation uses a fixed value for this threshold.

Step 7 – Reschedule. As network conditions can change or our initial estimates can prove wrong, we need to periodically re-evaluate our scheduling decisions. That's what we aim to achieve through the monitoring of real time system performance. We monitor conditions on both the wired and wireless segments and readjust our scheduling decision if the available bandwidth on either of these segments change. Section 4.3 provides details on our specific implementation for monitoring wired and wireless conditions.

Note that despite our best efforts our transfer finish times are not strict guarantees but educated guesses based on *current* conditions. As we show in our evaluation, Catnap is able to finish transfers on-time or with minimal delay in a variety of test scenarios. There could be other reasons why rescheduling may be required (e.g., arrival of new requests). As we allow new requests to be multiplexed with previously scheduled slots, there is no issue of *un-fairness* caused due to earlier scheduling decisions.

```
1. CALCULATE BATCH SIZE B
2. ESTIMATE MAX_WAIT_THRESHOLD
3. BATCH DATA UNTIL BATCH SIZE >= B OR
   NO PACKET FOR MAX_WAIT_THRESHOLD
4. BURST DATA
```

Figure 5: Basic steps performed in the batch mode.

3.5.2 Batch Mode

Figure 5 lists the basic steps performed by the batching algorithm. These steps are performed for each client that wants batched transfers. In step 1, we calculate the batch size that can ensure a certain sleep interval for the device, computed through Equation 1. Of course, our target sleep time also depends on the delay that can be tolerated by the user. For example, if the user is short of battery she may be willing to tolerate more delay, while in other scenarios she may not tolerate any additional delay. In the current design, the user gives a binary indication i.e., whether it wants to use the batch mode or not. If batch mode is desired, then the proxy uses a fixed pre-determined threshold for the maximum additional delay that a transfer can bear (set to a fixed value of 2 sec in the current implementation).

In step 2 we consider situations where we may never get enough data for batching. To address these cases, we estimate MAX_WAIT_THRESHOLD, which is the maximum interval of time we should wait for new data to arrive for a certain batch. If we do not get any packet for this long, we send the partial batch to the client. In our current implementation, we set this threshold to twice the RTT between the client and proxy – as no packet during this duration is a good indicator that no more data will arrive in the near future. The above two steps determine how long we batch data for a given client.

4. IMPLEMENTATION

We have implemented a prototype of Catnap in C for the Linux environment. It works as a user level daemon and implements the core design components of Catnap while using simple and convenient alternatives for modules that are not central to the working of Catnap. We give a brief overview of the important parts of the implementation.

4.1 Proxy

The proxy uses TCP on both the wired and wireless segments of the end-to-end path. It listens for client requests on multiple ports – there is one port for *all* Catnap-enabled applications and separate ports for every legacy application that is not Catnap enabled. Each incoming port has an associated forwarding rule that determines the destination of the request. For legacy applications, like HTTP and IMAP, the request is forwarded to the corresponding application proxy running on the middlebox. For new Catnap enabled applications, the client also sends information about the destination which is used by the proxy to forward the request to the appropriate server.

The proxy maintains separate state for each request. This includes the two sockets – one in each direction (proxy - server, proxy - client), a buffer associated with each socket, and other control information that helps in making scheduling decisions. There is a separate thread for each request that is responsible only for reading data from both the sockets and writing it to the buffer. The writing of data to sockets is handled by the scheduler.

4.2 Sleep Modes

Our implementation supports the use of two types of sleep modes: i) 802.11 PSM and ii) S3 mode. We enabled the PSM mode on the client device – as a result it goes into sleep state whenever it encounters an idle period longer than a pre-determined duration. The minimum value of this duration varies for different NICs and was generally found to be between 30ms to 250ms for many popular wireless NICs. In the sleep mode, the NIC wakes up at every beacon interval to check for incoming traffic. In summary, we used the standard PSM implementations supported by the NICs, without any Catnap specific modifications.[6]

Like the 802.11 PSM, we used standard mechanism of enabling S3 mode in client devices. In our current implementation, the client device goes into S3 state without notifying the user. However, in practice, we expect a proper user interface that can involve the user in this decision. In order to wake-up a sleeping client device, the proxy sends a Wake-On-LAN [2] magic packet to the client. Many recent wireless NICs do support the wireless WOL feature [3], but we did not have access to this hardware, so we used the wired ethernet interface for the magic packet. With the wireless WOL support, mobile devices will use their wireless interface in PSM mode to listen for magic packets from the Catnap proxy i.e., the client wireless NIC will wake up at every beacon interval to check for any magic packet from the proxy.

4.3 Scheduler

The centralized scheduler within the proxy is responsible for all scheduling decisions in both directions. We have implemented three types of schedulers: i) normal scheduler, which is not Catnap enabled, ii) Catnap-normal mode, and iii) Catnap-batch mode. On the wired side we use the normal scheduler while on the wireless side we use one of the two Catnap enabled modes. This implementation strategy allows us to use the same proxy at the client side as well. For example, we could have the client NIC sleep even longer if the client itself batched its own traffic towards the server.

Wired Bandwidth Estimation Module: This module assumes that the access link bandwidth is known and fixed (e.g., DSL). Future implementations can be extended to periodically use existing bandwidth estimation tools to estimate the access link bandwidth [17]. Once the transfer starts, this module keeps track of the throughput of the transfer – this allows identification of cases where the access link may not be the bottleneck (e.g., server being bottlenecked). This process is only applied to long transfers (> 1 sec) because their throughput information is likely to be more accurate and also because larger transfers have more impact on other ongoing transfers.

Wireless Bandwidth Estimation Module: Our wireless bandwidth estimation method is inspired by prior work that utilizes medium busy time information, along with the bit-rate used between the client and AP, to estimate the available capacity [16, 22]. The product of these two terms i.e., (bit-rate * medium free time), is often a good estimate of the capacity, as it captures the channel conditions as well as the overall load on the network. For the medium free time, we use a passive approach of overhearing data transmissions in monitor mode and calculating the medium busy time based on a moving 1 sec window. In this window, we calculate the sum of the bytes corresponding to the overheard frames and the transmission time of these frames (based on the bit-rate of the transmission). Note that we need to ignore the time that the proxy uses the medium for its *own* transmissions. The medium free time information is combined with the bit-rate information[7] – that can be retrieved from most 802.11 drivers – to get the available capacity.

Note that the above mechanism relies on looking at a window of time to decide on the available bandwidth. Naturally, this does not account for cases where an unexpected burst of traffic from another station overlaps with a scheduled time slot. The underlying fairness provided by 802.11 helps in reducing the negative impact of such cross traffic. In addition, we inflate the virtual time slot by 10% as padding time to deal with such unexpected cross traffic. In our evaluation, we show that this approach works well under a variety of cross traffic scenarios.

5. EVALUATION

The goal of our evaluation is to quantify the energy savings made possible by Catnap for different scenarios. Our evaluation can be divided into two parts. First, we consider several micro-benchmark scenarios related to Catnap's scheduling. In the second half of the evaluation, we evaluate the energy savings possible through Catnap on actual devices and for a variety of scenarios. Our experiments use a simple custom request-response application, which allows us to measure the benefits of Catnap without worrying about

[6]How to optimize the PSM implementation for Catnap is an interesting future research direction.

[7]The actual achievable throughput is lower because of some fixed protocol overhead of 802.11 and TCP.

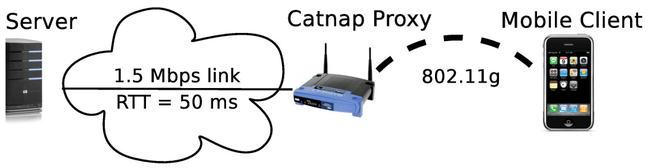


Figure 6: Network configuration used in the experiments.

the complexities of different application protocols. Section 6 presents two case studies that show how Catnap can be leveraged by legacy application protocols: HTTP and IMAP.

5.1 Evaluation Summary

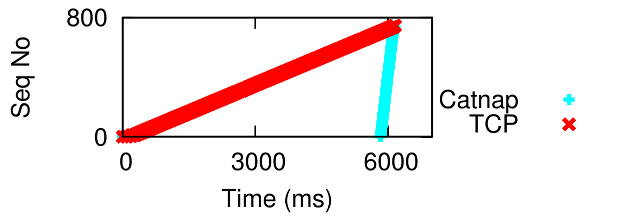
The key points of our evaluation are as follows:

- **Scheduling:** We show that the scheduler accurately estimates conditions on the wired and wireless segments, thereby ensuring maximum sleep time to clients with no added delay in transfer times. (Section 5.3.1)
- **Rescheduling:** We show how Catnap can efficiently address the challenge of rescheduling transfers. We consider several cases where rescheduling is required: i) server getting bottlenecked, ii) presence of wireless cross traffic, and iii) arrival of new requests at the proxy. (Section 5.3.2)
- **Energy Benefits:** Through experiments, we quantify the sleep time for both the wireless card and the device itself and how it translates into battery life improvement for real devices. Table 1 presents the summary of the evaluation. The benefits with Catnap get more pronounced as we increase the size of transfers. Specifically, Catnap allows the NIC to sleep for almost 70% of the time for a 5MB transfer while the device can sleep for around 40% of the time for a 10MB transfer. These energy savings translate into up to 2-5x battery life improvement for Nokia N810 and IBM Thinkpad T60. (Section 5.4)

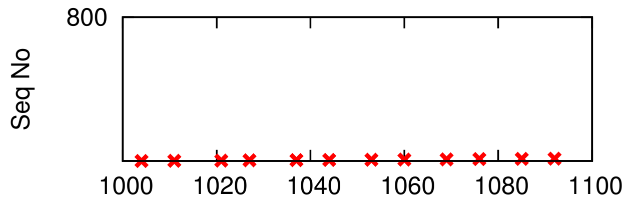
5.2 Setup

Figure 6 shows the network configuration used in the experiments; it represents a typical residential wireless access scenario. The wired WAN connection is emulated using the traffic control module of Linux. On the wireless side two different configurations are used. We used the Emulab wireless testbed for some of the experiments while for other experiments we used an actual AP connected to a Catnap enabled laptop via a 100Mbps connection. The main reason for using the Emulab testbed was the convenience of introducing cross traffic on the wireless segment (as it provides multiple wireless nodes), so we can evaluate the scheduler under different conditions.

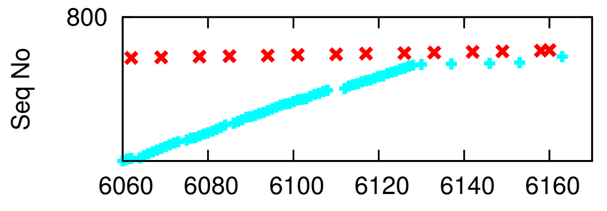
We used three client devices in our experiments. The first one is a Nokia N810 Internet Tablet which runs a lightweight Linux distribution. It has a 400Mhz processor with 128MB DDR RAM and also supports a 802.11g wireless interface with PSM capability. The tablet was running the default background processes during our experiments. The second device is a standard IBM thinkpad T60 laptop with



(a) Transfer Progress from start to end.



(b) Zooming into 1000-1100 ms period.



(c) Zooming into last 100 ms of transfer.

Figure 7: Single Transfer. Finish time is the same in both cases (a). Middle graph zooms into a middle period while the bottom graph zooms into the last 100ms of the transfer.

support for 802.11 PSM and Ethernet Wake-on-LAN. Finally, some of the experiments also involve an Emulab desktop node that is equipped with Wifi.

Our evaluation compares different modes of Catnap with standard end-to-end TCP-based transfers (TCP)[8]. Also, unless otherwise stated we present the mean of five runs, and the min and max values are within 5% of the mean.

5.3 Scheduler - Microbenchmarks

We conduct an evaluation of the efficacy of the scheduler using simple and realistic scenarios that a Catnap proxy will likely face. We first establish the accuracy of the scheduler and then show how it adapts to different situations by rescheduling transfers. All the results in this section show the sequence number of packets received by the client as a function of time.

5.3.1 Scheduler Accuracy

In this experiment the network conditions are stable and there is only one transfer. The main goal is to test the accuracy of the wireless bandwidth estimation module. Figure 7(a) shows that transfer times are the same for both Catnap and TCP. Figure 7(b) zooms into the middle period of the transfer. We see that with TCP each packet arrives after

[8]In our network configuration, the performance of end-to-end TCP and of using separate TCP sessions over the wired and wireless paths was similar.

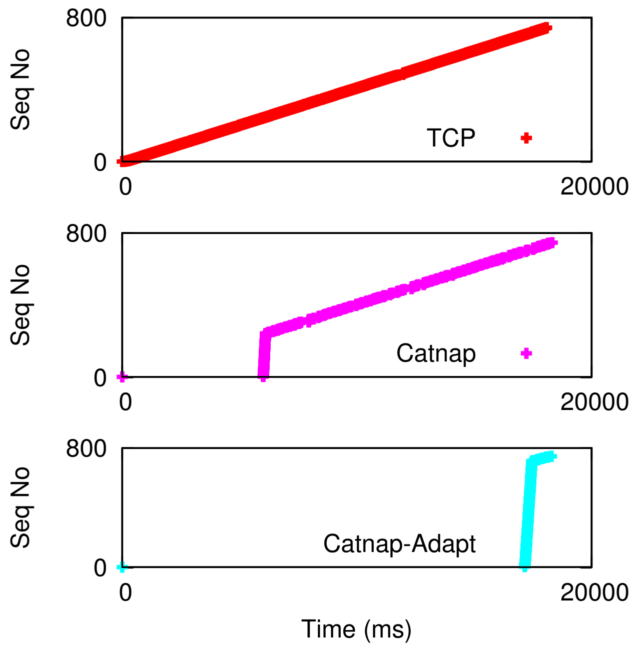


Figure 8: Reacting to a server being bottlenecked (500Kbps whereas access link is 1.5Mbps). Middle graph shows Catnap without adaptation – less batching due to estimation for earlier transfer completion. Last graph shows Catnap with adaptation.

a gap of a few milliseconds while Catnap is batching packets during this time. Figure 7(c) zooms into the last 100ms of the transfer. We see that Catnap is bursting packets to the client. The tail at the end shows Catnap's conservative approach that slightly inflates each virtual time slot to ensure that Catnap does not exceed the transfer deadline.

5.3.2 Adaptation and Rescheduling

We now consider several scenarios where adaptation is required.

Adapting to changes in server bandwidth: We now consider the case where the server becomes the bottleneck soon after the start of the transfer. In the experiment, the server gets bottlenecked at 500Kbps. Figure 8(b) shows the case where the Catnap proxy does not adjust to the lower server throughput and continues to consider the bottleneck bandwidth to be 1.5Mbps. As a result, it sleeps less because it expects the transfer to end soon. In contrast, adjusting to the lower server bandwidth allows approximately 200% more sleep time to the client (Figure 8(c)). Note that the adaptation is accurate enough to ensure that the transfer time does not increase. Finally, because we constantly monitor the progress of the transfer, we can adjust to the conditions at any point during the transfer.

Adapting to wireless cross traffic: We now consider different cases where we explicitly induce cross traffic on the wireless side. In the first experiment, we start a long-lived TCP flow (cross traffic) during the batching phase of the Catnap transfer. In this scenario, the wireless medium is heavily congested as the TCP background traffic uses a wireless bit rate of 1Mbps. Figure 9(c) shows that Catnap adapts to the cross traffic by rescheduling the time slot at

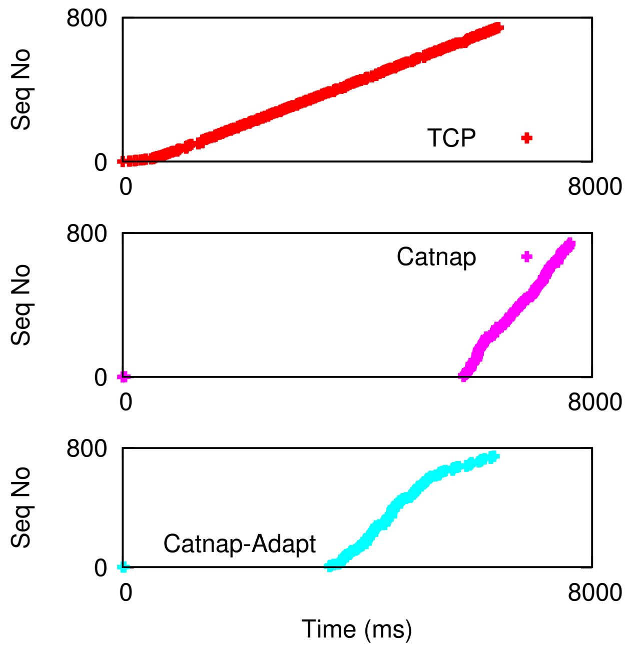


Figure 9: Reacting to TCP wireless cross traffic. Top graph shows TCP while middle one shows Catnap without adaptation. It anticipates more bw on the wireless side and therefore exceeds the transfer time while Catnap with adaptation is able to adjust (bottom graph).

an earlier time. Without such adaptation (Figure 9(b)), the transfer time is increased by around 15%.

We also conduct two experiments where the behavior of the cross traffic is less predictable i.e., ON-OFF behavior. In the *web-like* scenario, there is an OFF-period of 2 seconds after an ON period during which there are 5 transfers each of size 100kB. In the *Catnap-like* scenario, the cross traffic starts at the exact time as the Catnap proxy has scheduled the virtual time slot – in this case it is too late for the proxy to react. Both the above scenarios are more challenging cases because bandwidth estimation is less likely to work in scenarios where the cross traffic is bursty. We conduct 10 runs for these experiments, with each run starting at a random time.

	Mean	Max
Web-like Traffic	$<$ 2% (120ms)	5% (300ms)
Catnap-Like Traffic	8% (480ms)	18% ($\approx$1sec)

Table 4: Mean and Max increase in transfer times under two different background traffic scenarios. With web-traffic, there is hardly any impact on Catnap transfers. If another Catnap-like AP starts a burst in the background then transfer is delayed by 8% on average.

Table 4 summarizes the results for the above two scenarios. In the web-like scenario, there is hardly any impact on transfer times because of the light load of the cross traffic. As our bandwidth estimation considers the bytes transferred in a window of time, the OFF periods dominate the ON pe-

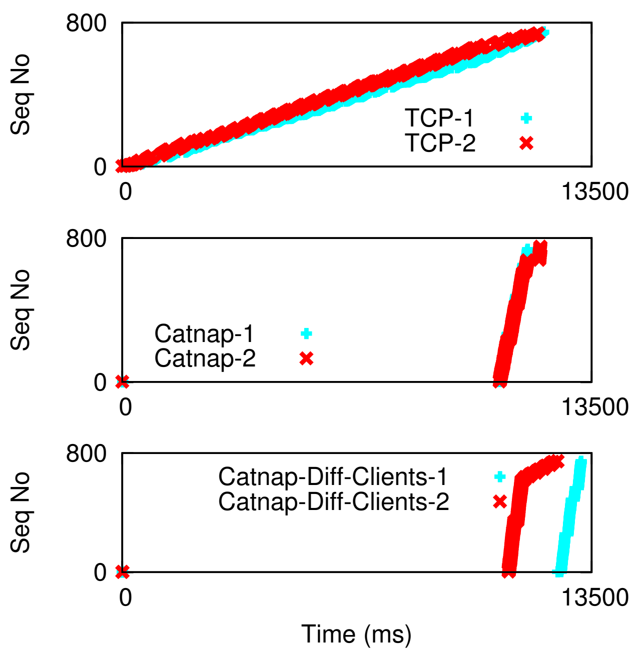


Figure 10: Two Overlapping Transfers. The middle graph shows multiplexing of time slots in Catnap if both the requests are from the same client (both transfers overlap so only one is visible) The bottom graph shows how requests from different clients will be allotted non-overlapping time slots in Catnap.

riods and therefore there is little impact on the available capacity. The worst case occurs when the Catnap time slot overlaps with the start of the ON period. However, as the results indicate, even the maximum increase in delay is just 5%. In contrast, the Catnap-like cross traffic can increase the transfer time by up to 18%. Again, the average increase is less (8%) because the extra padding with time slots allows us some leeway to deal with such unexpected cross traffic. These results show that, while we cannot provide a hard guarantee that the transfer time will not increase, we can minimize this chance and the impact is generally limited.

Adapting to multiple requests: We now consider scenarios where a Catnap proxy has to deal with multiple requests. We report the experimental results of the more challenging scenario where the virtual time slots of a future request overlaps with a previously scheduled transfer. Figure 10 shows the results of two overlapping requests. TCP multiplexes the two requests naturally whereas Catnap can handle them in two different ways (see Figure 4). Figure 10(b) shows the case where the requests are from the same client and are combined in a bigger time slot, while ensuring that transfer times do not increase. In contrast, Figure 10(c) shows the case where the requests are from different clients. In this case, we assign non-overlapping slots to both clients, which reduces the up time of *both* clients. However, it results in increased transfer delay for the second transfer – if this delay is not acceptable then we multiplex the two requests (similar to Figure 10(b)).

5.3.3 Discussion

The above experiments show that Catnap's bandwidth estimation is accurate and allows the scheduler to adapt to changes in network conditions as well as handle multiple requests that may overlap with each other. The results also show that variations on the wireless side are less critical as long as the available bandwidth on the wireless side is significantly higher than the bottleneck link bandwidth.

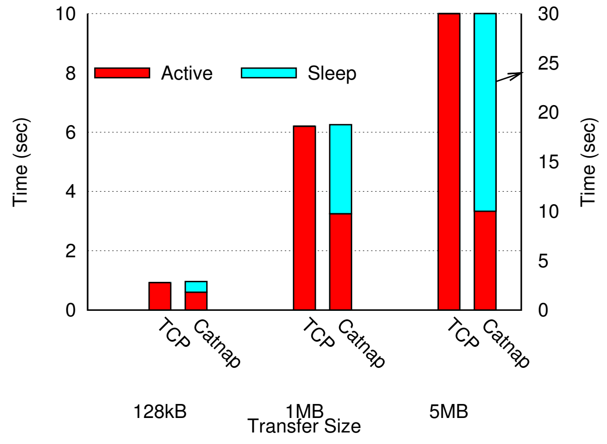


Figure 11: NIC Sleep time for Nokia N810. Catnap allows even 1 second long transfers to sleep, while leading to the same overall transfer completion time across all tested transfer sizes.

5.4 Energy Savings

We conducted several experiments to characterize the potential energy benefits of using Catnap. We have quantified both 802.11 PSM savings as well as savings using S3 mode. For the N810 platform we focus on PSM savings as network activity is the major power consumer and S3 mode does not offer much additional savings. In contrast, for T60 we focus only on S3 mode as PSM alone does not offer any worthwhile power savings.

5.4.1 NIC PSM Energy Savings

First, we want to evaluate how well Catnap performs in terms of energy benefits by allowing the NIC to sleep in-between transfers. For the experiments, we used the deepest PSM mode available in N810, which offers the greatest energy savings and enters sleep mode if there is an idle period of 200ms (we call this `psm_cost`).

How long can the NIC sleep. We evaluated the transfer time and the proportion of transfer time spent sleeping for TCP and Catnap. Figure 11 shows this comparison for different transfer sizes. Catnap amortizes the `psm_cost` over the total transfer time and therefore performs better as the transfer duration increases. As we can see from the figure, the transfer time does not increase with Catnap and yet it is able to sleep for around 30% of the time for a 128kB transfer, which is roughly the size of many popular web-pages. Furthermore, it can sleep for around 50% of the time for a 1MB transfer, which is roughly the pre-buffering requirement of a `you-tube` video, and around 70% of the time for a 5MB transfer, which is the size of a typical mp3 song.

Does increased NIC sleep time improve battery life? Here we want to quantify how increased NIC sleep times with Catnap translate into battery life improvement for the N810. This experiment uses two different workloads:

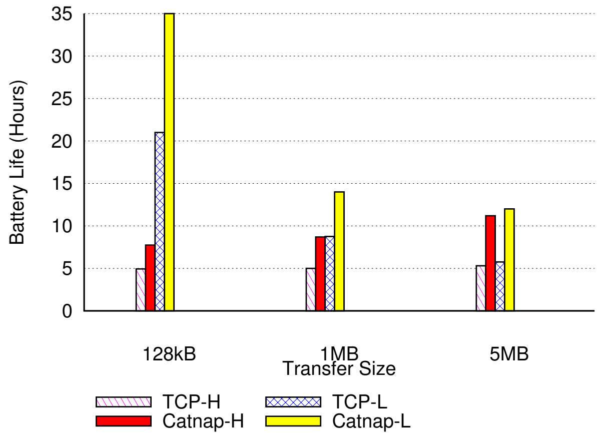


Figure 12: Battery life improvement for N810 due to NIC savings under heavy (H) and light (L) workloads.

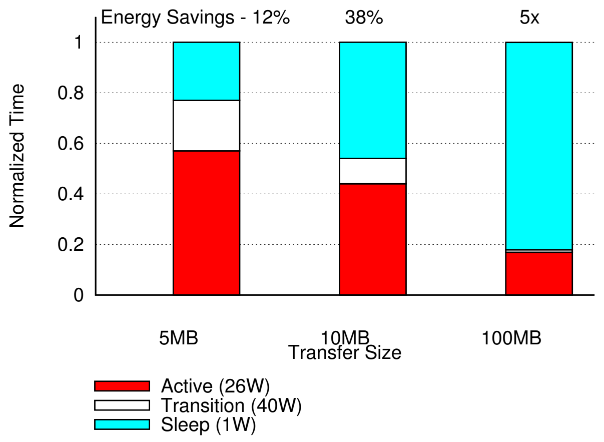


Figure 13: Catnap's transfer time and energy costs using S3 mode (normalized wrt TCP) for T60. TCP based transfers remain in active state for the whole duration.

i) heavy load (H) where transfers are made continuously, and ii) light load with an idle time of 5 seconds between transfers. We evaluate how long the battery lasts by fully charging it before the experiment and letting the experiment continue until the battery power completely drains out. The N810 automatically switches off the display back-light if there is no human activity, so all reported results are under this setting.

As Figure 12 shows, greater sleep opportunities in Catnap translate into significant battery life improvement over TCP. Our results confirm the N810 specifications that the device consumes negligible power in idle state. As a result, we notice that NIC power savings are beneficial for both heavy and light workloads, with up to 2x improvement in battery life.

5.4.2 Deep Sleep Mode (S3) Energy Savings

We wanted to evaluate the energy benefits of using Catnap with S3 mode for a laptop. As we noted earlier, S3 mode takes roughly 10 seconds to suspend and resume back, and is therefore only suitable for longer transfers. Future reduction to the S3 mode overhead could possibly make it useful even for smaller transfers. Note that Catnap just creates more opportunities to use S3 mode, so issues such as how to deal with incoming traffic when the client device is in S3 mode are orthogonal and can be addressed using existing mechanisms [4].

For this experiment, the client makes a request and then enters S3 mode. The proxy continues with the transfer, caches the data in its storage, and at the appropriate time wakes up the client by sending a wake-on-lan magic packet. The client wakes up and downloads the cached data from the middlebox. Figure 13 shows the normalized transfer times with respect to baseline TCP transfers for different file sizes. It also shows the proportion of time Catnap spends in each of the three states: Active, Transition, and Sleep, in addition to the power consumed in these three states. Note that TCP transfers remain in active state for the whole duration but Catnap allows significant sleep time without increasing the total time to complete a transfer. More specifically, even for a 5MB transfer that only lasts for 30 seconds, Catnap is able to spend around 20% of the time in S3 mode while for larger transfers (100MB) the proportion of sleep time increases to around 85%, which translates into 5x energy savings. If files are constantly downloaded, these energy savings correspond to up to 5x improvement in battery life.

6. CASE STUDIES

The goal of the case studies is to gain some understanding of how different applications can benefit from Catnap. We pick two popular application protocols: IMAP and HTTP. Our goal is to make these protocols work with Catnap without modifying standard clients or servers. Furthermore, we are interested in evaluating the energy benefits when these applications use Catnap. Specifically, we focus on the challenging cases of short web transfers and small email message exchanges, which require the use of Catnap's batch mode in order to get energy benefits.

	Catnap-batch	TCP
Start-up Phase (≈ 3 sec)	Same	Same
Download Phase (3.6 sec)	650 ms	3.6 sec

Table 5: Comparison of NIC up time for an email session involving 20 messages of 20kB each. In the start-up phase, both TCP and Catnap-batch perform the same but in the download phase, Catnap-batch results in around 80% less NIC up time.

.

6.1 IMAP

We enabled the use of standard IMAP clients and servers with Catnap. This required the use of a modified IMAP proxy at the middlebox that parsed the responses from the IMAP server and added the length information as a hint for the Catnap proxy. The modification process was simple and required approximately 50 additional lines of code. All the features that were supported by the IMAP proxy worked fine with Catnap.

We observed that the IMAP behavior in practice is more complex than a simple request-response protocol. Broadly, there are four steps that a typical client follows in downloading email messages from the server: authentication, checking inbox status, downloading message headers, and downloading message bodies. We refer to the first three steps as the *start-up* phase and the last step as the *download* phase. Most clients only download the body when it is required, which means that for every message there is a separate request to download it. However, this option can be changed by setting the client to work in *offline* mode where the client downloads all unread messages.

The start-up phase involves the exchange of short control messages that are not large enough to get energy benefits. In the download phase, if the client is downloading large messages (e.g., attachments) then the cost of the start-up phase gets amortized and we observe energy benefits that are similar to our results in the previous section. However, most messages that do not have attachments are smaller than 100kB, which makes it difficult to get any energy benefits if we treat each message as a separate ADU. Here, we focus on how we can get energy benefits for small messages by batching them using Catnap's batch mode.

We conduct a simple experiment where we consider a traveller who has multiple unread messages to download (e.g., a professor checking her e-mail at the airport on her way to a conference). We consider the case when the email account has 20 unread messages of 20Kb each. This is a typical size of a call for paper announcement or an email belonging to a discussion thread on a mailing list. We compare the NIC up-time for Catnap's batch mode with that of using TCP. Table 5 shows that in the start-up phase, the up time is the same for both the schemes because there is not enough data for the batch mode to get any real benefit. However, once the download phase starts, the NIC is only active for 650ms with Catnap's batch mode whereas it is up for 3.6 seconds with TCP. This shows that if there are more messages – even if they are individually small – then the batch mode can combine them to get energy benefits.

6.2 HTTP

Catnap with web browsers and servers: As a first step we focused on using Catnap with standard web browsers and servers. We installed a modified squid proxy on the middlebox – the Catnap proxy interacted with the squid proxy rather than with servers. The modification to the squid proxy involved adding the Catnap hint (i.e., transfer length) that is used by the Catnap proxy for scheduling. Adding this hint was simple as HTTP responses from servers already contain the length of the responses in the HTTP header.[9] The whole modification process involved the addition of 20 lines of code. We verified that the client browser could view different kinds of web-pages (youtube, cnn, static and dynamic pages, pages with SSL, etc) while using Catnap. Note that we could make the same changes to the web server, eliminating the need for the web proxy.

Why smaller web pages may not benefit. Typical HTTP protocol interactions between the browser and server are much more complex than a simple request-response exchange. A webpage has many embedded objects and they are usually fetched using separate requests over different TCP connections. We observed that for large transfers the cost of these multiple requests gets amortized and the energy savings were similar to the simple request-response protocol. However, for normal web browsing, energy benefits with Catnap were minimal. Table 6 provides an insight into the reasoning for this behavior. While most web-pages as a whole are big enough to benefit from Catnap, individual objects are usually smaller than 100kB, which makes it difficult to get any energy benefits. This indicates that we should treat the web page as one large block of data rather than separate small objects.

[9]Even dynamic content is sent as a chunk in HTTP and the response contains the length of the chunk.

	Total Size	Max Size Object
www.google.com	36kB	15kB
www.yahoo.com	540kB	48kB
www.amazon.com	580kB	60kB
www.nytimes.com	860kB	260kB
www.cnn.com	920kB	150kB

Table 6: Comparison of different popular web-pages in terms of total size and the size of the largest embedded object within that page. Individual objects are generally too small (median object size of these web-pages is also less than 50kB) to get any energy benefits but the total size for most pages is large enough to get energy benefits from Catnap.

Catnap Batch mode: We wanted to evaluate the effectiveness of Catnap's batching mode for web-pages that have multiple embedded objects but no scripts. In such cases, the browser first makes a request for the main page and after it gets the response, it makes parallel requests for all the objects embedded within the page.

	Catnap-batch	TCP
Download Time	1.5 sec	1 sec
NIC up time	650 ms	1 sec

Table 7: Comparison of download time and NIC up time for a small web-page download. Catnap batch mode results in 35% less NIC up time compared to TCP.

We conduct a simple experiment by selecting a web-page with the above characteristics (`www.cs.cmu.edu/~bmm`). The objects within the page are individually all smaller than 100kB but their combined size is close to 150kB, which is large enough to get energy savings. Table 7 shows the download time and the NIC up time for Catnap-batch mode and TCP. Note that Catnap's normal mode performs the same in this case as TCP because the objects are individually too small to get any energy benefits. Without Catnap it takes 1 sec to download the web-page and the NIC is active for the whole duration. With the use of Catnap-batching, the download time increases to 1.5 sec, however, the NIC is active for only 650ms, which shows a reduction of 35% in up-time for the NIC. In case of N810, which hardly consumes any power when the NIC is sleeping, the NIC savings translate into equivalent system level savings. However, because we are increasing the total time of transfers, devices that consume significant power even when their NIC is sleeping may not get similar benefits from the batch mode.

Embedded Scripts and Application Specific Techniques: In practice many web-pages have scripts that further complicate the HTTP protocol as browsers make further requests only after executing the script. The script execution can itself generate requests for new objects, making it difficult to batch large amount of data. Researchers have proposed *application specific* approaches that can turn the HTTP protocol into a single request-response exchange [28]. The general approach is to have the proxy respond with all the embedded objects (including embedded scripts), in response to a request for a web-page. Using such techniques at the middlebox can extract maximum benefits out of Catnap as they turn the complex protocol interaction between the client and server into a simple protocol where the client makes a request for the main page and receives all the relevant objects in response. This could further improve the energy savings with Catnap. For example, for the above webpage experiment, the NIC up-time can be reduced to 450 ms if we use application specific support at the middlebox.

7. USER CONSIDERATIONS

Increased Delay: Most of our evaluation focuses on the case where Catnap does not add delay to transfers. However, increases in delay are possible, for example, when using Catnap's batch mode or when there is a sudden surge in load on the wireless network. Our results show that the additional delay is small – of course, the users can even avoid this delay if they do not want the extra energy savings. So there is clearly a trade-off that users need to consider while deciding on whether they want to use the batch mode or not.

Interface: Besides possible changes in delay, Catnap may also affect the interface of some applications. The most important interfacing issue is how to expose the use of S3 mode to the user. Most devices already have sensors to detect whether a user is currently using the device or not. We can leverage this support even though the user needs to be ultimately involved in the decision to enter the S3 state. This could be implemented in the same way as automatic reboots are implemented for system updates i.e., allowing the user to explicitly cancel the action within a certain time limit.

In addition to the S3 mode, there could be other cases where the interface to the application needs to be changed. For example, for large downloads, users expect a status bar that shows the steady progress of the transfer. With Catnap the transfer stalls at the start and finishes with a rapid burst at the end. Despite such a change, the interface could certainly expose the predicted completion time as estimated by the scheduler, thus making the user experience more predictable.

Control: Note that many mobile devices already provide users with power-save options (e.g., dim screen, reduce processor frequency) that affect the user experience. Catnap naturally fits this model. The user interface for setting Catnap preferences (e.g., use of batch mode, tolerance to extra delay, use of S3 mode) could be integrated with existing power save applications. This will raise interesting questions of how users address the trade-off between improved battery life and possible increase in transfer time.

8. FUTURE WORK

We now discuss two possible extensions to Catnap.

UDP-Blast: Catnap can potentially sleep more if it could send data faster on the wireless segment. Our current evaluation shows results for TCP over standard 802.11, but several optimizations are possible. Most of TCP's reliability and congestion control mechanisms are redundant for the wireless segment in a typical home setting. Similarly, standard 802.11 has noticeable overheads that limit throughput. Instead we can use what we term "UDP-Blast". UDP-Blast is a UDP-based protocol that runs over 802.11e block transmission mode. Our initial results show that UDP-Blast shortens the up-time of the NIC by another 30% by achieving significantly higher throughput than TCP over standard 802.11. Note that in an end-to-end transfer (without batching at the middlebox), the 802.11e mode is not useful because there is not enough data to send. We intend to add flow control and light weight reliability to UDP-Blast so that it can be used as a replacement for TCP on the wireless side.

Beyond energy savings: Though we focused on energy-related benefits, Catnap can also help users in other ways. One example is providing flexible mobility options. Consider a user who wants to download a 700MB movie in the morning. Assume that the download should take around 30 minutes. Without Catnap, the user device has to remain connected for the whole duration, which means that the user cannot take the device to her office. With Catnap, the user can issue the download request, un-plug the device, and go to office. Meanwhile, the middlebox continues to download and cache the movie locally (possibly in a supplemental USB storage device). When the user returns home, she can download the movie in 3 minutes from her wireless middlebox. This opens a plethora of pre-fetching opportunities: the usually idle wired connection can be used to pre-fetch data at the middlebox and the fast wireless connection can later be used to burst the data to the user device.

9. RELATED WORK

The problem of improving energy efficiency of mobile devices has received considerable attention in the research community. Here, we discuss the work that is most relevant to Catnap.

Exploiting Idle Opportunities: There have been several proposals to exploit idle opportunities that are *already present* at different levels. Most of these proposals are orthogonal to Catnap as they target a different application or network scenario. For example, there are proposals that target idle time between web requests or during TCP slow start [20]. Similarly, some proposals focus on interactive applications [23, 5], or consider power savings for sending data over a GPRS link [28]. Catnap targets scenarios where applications are constantly sending data but there are idle opportunities due to the network characteristics i.e., bandwidth discrepancy between wired and wireless segments.

A recent proposal by Liu and Zhong [21] is most relevant to Catnap as it considers similar network settings. It exploits advances in modern 802.11 transceivers to enter low overhead power saving states during the small idle intervals between packets. Catnap takes a different approach: it *combines* these small gaps between packets into larger sleep intervals for applications that consume data in blocks – as a result, Catnap can exploit deeper sleep modes such as 802.11 PSM and S3 mode. We believe that future mobile devices will use both these techniques: for data oriented applications, aggressive sleep approaches, like Catnap, are more suitable, while for interactive applications, exploiting low power saving states becomes important.

Traffic Shaping: There is a significant body of work that uses traffic shaping in order to *create* more sleep opportunities for both the wireless NIC and the system [26, 13, 9, 24]. Most of these schemes shape traffic at the end-points, in an application specific manner [9, 26]. Nedveschi et al.[24] also consider traffic shaping within the network core, but at the expense of added delay for all application types. Unlike these proposals, Catnap shapes the traffic considering the bandwidth discrepancy between the wired and wireless segments. As a result, the traffic shaping can only be done at the middlebox, rather than the end-points. Furthermore, Catnaps takes into account the challenges of scheduling and rescheduling transfers on the wireless segment – such challenges are not considered by existing proxy based solutions [27]. Finally, as Catnap uses workload hints, it can use sleep modes for long duration of time without affecting user experience.

Deep Sleep Modes: The S3 mode for Catnap is similar to the use of deep sleep modes in Somniloquoy [4] and Turducken [29]. The common theme is that the main system sleeps while another device keeps up with the data transfer. In case of Catnap, this device is the wireless AP while the other two systems rely on additional low power hardware that is co-located with the mobile device. Avoiding the introduction of additional hardware on the mobile device, however, comes with its own challenges since we now have to intelligently schedule transfers on the wireless segment. Another difference is that the aforementioned prior systems use a pure store and forward approach which increases the duration of a transfer. In contrast, Catnap uses a cut through approach where the transfers over the wired and wireless segments are overlapped in order to keep the transfer time the same. However, this requires workload hints from applications, which is not required in the other systems.

Hints and Batch Mode: Many design decisions made in Catnap are inspired by previous work in the area of energy management for mobile devices. Anand et al. [5] use application *hints* to determine the intent of applications while using the NIC, which helps in keeping the NIC active for interactive applications. However, for data oriented applications that constantly send data, their approach always keeps the NIC active. Catnap also uses hints from applications but the hint relates to the length of the transfer. Similarly, Catnap's batch mode is based on the principle of trading application performance for improved battery life – this principle is well-understood, for example, in the context of lowering the fidelity of multimedia information to improve energy consumption [15]. Catnap applies this principle in the context of increasing the transfer delay to get energy benefits. More recently, TailEnder explores the use of batching multiple requests at the client to improve energy efficiency of outgoing transmissions [7]. In contrast, Catnap's batching mechanism has broader applicability as it can be used at both the proxy and the client, thereby allowing batching of both incoming and outgoing data at the client.

Discussion: In summary, Catnap's uniqueness comes from the following: i) unlike most work on energy management, Catnap focuses on periods of peak network activity when applications are constantly sending data, ii) it leverages a range of sleep modes (NIC PSM, S3) rather than focusing on a specific one, and iii) it uses ADUs for application independent and yet application aware energy savings. Catnap therefore occupies a unique space in the energy saving realm: it provides an alternate view of how data transfers should be conducted over the Internet to maximize power savings for mobile clients.

10. CONCLUSION

In this paper, we presented the design, implementation, and evaluation of Catnap – a system that improves battery life of mobile devices by exploiting the bandwidth discrepancy between wired and wireless segments. Catnap is implemented as an application independent proxy to decouple wired and wireless segments, and schedules ADU transmissions on the wireless side using hints regarding the application workload. Catnap also considers changes in network conditions, as well as arrival of future transfer requests, thereby addressing the key challenges associated with scheduling/rescheduling of data transfers. We show that Catnap offers several strategies to reduce energy consumption, which accommodate a variety of applications, and provide benefits for a range of mobile devices. Specifically, for medium to large transfers, it allows both the NIC and the device to sleep for a significant fraction of total transfer time, resulting in battery life improvement of up to 2-4x for Nokia N810 and Thinkpad T60. Finally, we believe that the benefits of Catnap will get even more pronounced in future with the continued growth of data intensive applications and further increase in wireless access bandwidth.

Acknowledgements

Dave Andersen, Ihsan Qazi, Amar Phanishayee, M. Satyanarayanan, Vyas Sekar, and Srinivasan Seshan provided useful feedback on earlier versions of the paper. M. Satyanarayanan also loaned out the Nokia N810 that was used for the experiments. Jason Hong gave valuable feedback on usability aspects of the batch mode. Finally, Alex Snoeren, our shepherd, and the anonymous MobiSys reviewers provided detailed and critical feedback that helped us in improving the presentation of this paper. We thank them all. This research was funded in part by NSF under award number CNS 0855137 and by an HEC fellowship award to Fahad R. Dogar.

References

[1] Linksys 350n router. `http://www.ubergizmo.com/15/archives/2006/10/linksys_wrt350n_gigabit_80211n_router.html`.

[2] Wake-on-lan. `http://en.wikipedia.org/wiki/Wake-on-LAN`.

[3] Wake-on-wireless lan. `http://www.intel.com/support/wireless/wlan/sb/CS-029827.htm`.

[4] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *NSDI'09*. USENIX Association, 2009.

[5] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *MobiCom '03*, New York, NY, USA, 2003. ACM.

[6] A. V. Bakre and B. Badrinath. Implementation and performance evaluation of indirect tcp. *IEEE Transactions on Computers*, 46(3):260–278, 1997.

[7] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for

Network Applications. In *Proc. ACM IMC*, November 2009.

[8] L. S. Brakmo, D. A. Wallach, and M. A. Viredaz. μsleep: a technique for reducing energy consumption in handheld devices. In *MobiSys '04*, pages 12–22, New York, NY, USA, 2004. ACM.

[9] S. Chandra and A. Vahdat. Application-specific network management for energy-aware streaming of popular multimedia formats. In *USENIX Annual Technical Conference*, Berkeley, CA, USA, 2002. USENIX Association.

[10] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM*, New York, NY, USA, 1990.

[11] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *IMC '07*, New York, NY, USA, 2007. ACM Press.

[12] F. R. Dogar and P. Steenkiste. Segment based internetworking to accommodate diversity at the edge. *CMU-CS-10-104*, 2010.

[13] C. fabiana Chiasserini and R. R. Rao. Improving battery performance by using traffic shaping techniques. *IEEE Journal on Selected Areas in Communications*, 19:1385–1394, 2001.

[14] K. Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03*, pages 27–34, New York, NY, USA, 2003.

[15] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP '99*. ACM, 1999.

[16] D. Gupta, D. Wu, P. Mohapatra, and C.-N. Chuah. Experimental comparison of bandwidth estimation tools for wireless mesh networks. In *IEEE INFOCOM*, 2009.

[17] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. In *SIGCOMM '02*, New York, NY, USA, 2002.

[18] S. Kandula, K. C.-J. Lin, T. Badirkhanli, and D. Katabi. FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput. In *NSDI*, San Francisco, CA, April 2008.

[19] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *WMCSA '02*, pages 40–48, Washington, DC, USA, 2002. IEEE Computer Society.

[20] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless web access using bounded slowdown. In *MOBICOM*, Atlanta, GA, 2002.

[21] J. Liu and L. Zhong. Micro power management of active 802.11 interfaces. In *MobiSys '08*, pages 146–159, New York, NY, USA, 2008. ACM.

[22] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill. Designing high performance enterprise wi-fi networks. In *NSDI'08*, pages 73–88, Berkeley, CA, USA, 2008. USENIX Association.

[23] V. Namboodiri and L. Gao. Towards energy efficient voip over wireless lans. In *MobiHoc '08*, pages 169–178, New York, NY, USA, 2008. ACM.

[24] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *NSDI'08*, pages 323–336, Berkeley, CA, USA, 2008. USENIX Association.

[25] T. Pering, Y. Agarwal, R. Gupta, and R. Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *MobiSys*, New York, NY, USA, 2006.

[26] C. Poellabauer and K. Schwan. Energy-aware traffic shaping for wireless real-time applications. In *RTAS '04: Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, page 48, Washington, DC, USA, 2004. IEEE Computer Society.

[27] M. C. Rosu, C. M. Olsen, C. Narayanaswami, and L. Luo. Pawp: A power aware web proxy for wireless lan clients. In *WMCSA '04*, 2004.

[28] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding. Cool-tether: energy efficient on-the-fly wifi hot-spots using mobile phones. In *CoNEXT '09*, New York, NY, USA, 2009. ACM.

[29] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: hierarchical power management for mobile devices. In *MobiSys '05*, pages 261–274, New York, NY, USA, 2005. ACM.