AppATP: An Energy Conserving Adaptive Mobile-Cloud Transmission Protocol

Fangming Liu, *Member, IEEE,* Peng Shu, John C.S. Lui, *Fellow, IEEE, ACM*

Abstract—Many mobile applications require frequent wireless transmissions between the content provider and mobile devices, consuming much energy in mobile devices. Motivated by the popularity of prefetch-friendly or delay-tolerant apps (e.g., social networking, app updates, cloud storage), we design and implement an application-layer transmission protocol, AppATP, which leverages cloud computing to manage data transmissions for mobile apps, transferring data to and from mobile devices in an energy-efficient manner. Measurements show that significantly amount of energy is consumed by mobile devices during poor connectivity. Based on this observation, AppATP adaptively seizes periods of good bandwidth condition to prefetch frequently used data with minimum energy consumption, while deferring delay-tolerant data during poor network connectivity. Using the stochastic control framework, AppATP only relies on the current network information and data queue sizes to make an online decision on transmission scheduling, and performs well under unpredictable wireless network conditions. We implement AppATP on Samsung Note 2 smartphones and Amazon EC2. Results from both trace-driven simulations and extensive real-world experiments show that AppATP can be applied to a variety of application scenarios while achieving 30%-50% energy savings for mobile devices.

Index Terms—Mobile cloud computing, energy efficiency, transmission management, stochastic optimization.

1 Introduction

With an increasing popularity of mobile devices in recent years, users are gradually shifting their preferences from traditional cell phones and laptops to smartphones and tablets. As indicated by Cisco [1], traffic from mobile devices is anticipated to exceed that of wired devices by 2014 and to account for 61% of the entire IP traffic by 2016. The prosperity of mobile markets is largely driven by rich-media applications—over two thirds of the global mobile traffic will be rich-media content including video, image and audio by 2017 [1]. The statistics from Flurry [2] further confirm that the time users spend on mobile applications has surpassed that of web browsing on both desktop and mobile devices since 2011, while users spend 80% of their time on rich-media apps such as social networking, gaming, news feed and videos.

To meet the demand for ubiquitous access to rich-media content, many developers have started to leverage cloud computing to overcome resource constraints on mobile devices. Mobile cloud computing, is emerging as a new computing paradigm which has fostered a wide range of exciting rich-media applications (e.g., Instagram, Viddy, Siri). However, these mobile-cloud applications require heavy data transmissions, which consume a significant portion of the mobile device battery through the use of its network interface [3]. The transmission burden is further exacerbated by cloud-based data backup and mobile-cloud storage services [4] such as Dropbox, which stores and synchronizes mobile data in Amazon S3 storage system. In fact, according to Cisco [1], mobile-cloud traffic is projected to constitute 84% of the entire mobile traffic in 2017, growing by 14 folds from 2012, which increases at a faster pace than battery capacity. It is a great challenge as well as an opportunity to manage the energy-efficiency of such a huge amount of data transmissions between the cloud and mobile devices.

In this paper, we propose *AppATP*, an *App*lication-layer *A*daptive *T*ransmission *P*rotocol targeting at energy-efficient data transfers between mobile devices and the cloud platform. The effectiveness of AppATP is hinged upon two observations: *First,* the energy consumption in transmission is largely affected by the intrinsic stochastic nature of wireless networks. This is especially true due to the instability of wireless connections and the fluctuation of communication bandwidth. Measurements show that more energy is consumed during bad connectivity, while less energy is consumed with good connectivity in mobile devices [5]. *Second,* many mobile apps are delay-tolerant, such as music/video download and data backup, while many other tasks are prefetch-friendly, for example, content retrieval in social networking services such as Twitter and Weibo[1].

Analogical to modern cloud-based paradigm such as Dropbox and Netflix, which utilize cloud computing to provide storage and video services, AppATP also leverages the computation and storage capabilities of

• *F. Liu and P. Shu are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China. E-mail:* `fmliu@hust.edu.cn`.
• *John C.S. Lui is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. E-mail:* `cslui@cse.cuhk.edu.hk`.

Manuscript received July XX, 2014; revised January XX, 2015.

1. Note that AppATP mainly focuses on, and explicitly intends to be used by, those mobile apps that are prefetch-friendly or delay-tolerant, rather than real-time or delay-sensitive apps.

cloud computing, yet with a *different objective* to reduce energy consumption for mobile devices. To some extent, AppATP utilizes cloud resources to provide transmission management and energy saving services. It buffers data of multiple apps at the cloud side before transmitting them to mobile devices, in order to judiciously manage and schedule data transmissions for multiple mobile apps. This is suitable for a wide range of delay-tolerant or prefech-friendly types of rich-media apps. The unique advantage of AppATP is to intelligently choose the *right timing* (periods of high bandwidth) for mobile devices to prefetch frequently desired data in apps like social networking services (e.g., Twitter, Weibo) and newsfeed, while deferring delay-tolerant data until good network condition arises in such applications as software/app-updates and content downloading.

Using the stochastic optimization framework, we propose a low-complexity and low-overhead online algorithm to decide whether a wireless connectivity is energy-efficient for transmitting data, without having to predict future bandwidth conditions. Note that we do not set fixed bandwidth thresholds to determine whether the current wireless connectivity is “good” or “bad” for energy-efficient transmissions, but adaptively make the transmission decision to strike a balance in the *energy-delay tradeoff* for multiple apps. Our protocol achieves reduced energy consumption with bounded delays.

Different from previous theoretical works (e.g., [5]), we have implemented and deployed AppATP on Amazon EC2 and Samsung smartphones running Android Jelly Bean OS. We conduct real experiments for social networking service (SNS) apps and file downloading, which utilize AppATP for energy-efficient prefetching and downloading. Results show that AppATP can achieve an energy consumption reduction of 30%-50% for mobile devices in various scenarios.

The remainder of this paper is organized as follows. In Sec. 2, we motivate the AppATP transmission protocol and discuss our design objectives and choices. In Sec. III, we describe system model and formulate our energy minimization problem. Sec. IV presents our energy-efficient transmission scheduling algorithm, along with its performance analysis. In Sec. V, we describe the implementation of AppATP with practical improvements, followed by extensive performance evaluation in Sec. VI. We discuss related work and conclude the paper in Sec. VII and Sec. VIII, respectively.

2 Motivation and Design

The energy consumption of mobile-cloud data transmission critically depends on wireless network conditions, which not only vary as users move, but also fluctuate depending on weather, building shields, congestion, etc. Such stochastic characteristics of wireless network condition cause unpredictable energy consumption in mobile-cloud communications: Measurement studies [6], [7] show that the energy consumption for transmitting a

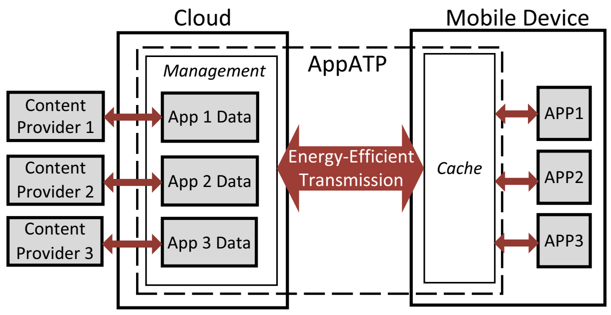


Fig. 1: An architectural overview of the AppATP framework.

fixed amount of data is *inversely proportional* to the available bandwidth. The reason is that the faster a user can download, the less transmission time is needed, and thus less energy is consumed [8]. Note that this calculation does not include the screen-on time if the user is actively waiting for the download process to complete — the screen drains power even faster while a user is waiting. This observation implies that transmitting data in good connectivity could save energy considerably compared to trasmitting data during bad connectivity.

Inspired by the considerations mentioned above, energy-efficient mobile-cloud communications can be achieved by seizing the “right” timing for data transfers. We therefore envision a transmission protocol for mobile apps that can automatically perceive network conditions and intelligently schedule data transmissions for different apps based on such conditions. Due to the energy saving and potentially downloading data, such a protocol can lead to prolonged battery life and enhanced user experience in mobile devices.

Admittedly, intentionally choosing the time to transmit data may incur delays and affect user experience in real-time or delay-sensitive apps. Hence, our design of AppATP mainly focuses on, and explicitly intends to be used by, those mobile apps that are prefetch-friendly or delay-tolerant, in which data transfers can be scheduled flexibly while not degrading user experience[2]. For example, in SNS apps like Facebook and Twitter, which account for 26% of the total app usage time [2], a user may not immediately check her account when her friends share something, and there is often a time interval between the current and next checking points. Thus, there is an opportunity to prefetch the desired SNS content during good network connectivity. User-generated photos and pictures such as those from Instagram and Viddy constitute a large portion of mobile traffic. Deferring the upload of such content for a short period may not hurt user experience. Furthermore, the synchronization of newly generated mobile data in Dropbox or the downloading of app software updates can also be deferred until good bandwidth is available,

2. Since prefetching has well-known problems of (1) causing wastage of bandwidth if the prefetched data is not consumed by the user, and (2) users may consume outdated data, we will discuss these issues in the cache management of mobile device in Sec. 5.2.

since the synced data may not be immediately requested by other Dropbox clients, and most people are not that sensitive to the update time of their apps. In all the cases mentioned above, AppATP can help flexibly schedule data transfers to conserve energy.

Note that AppATP only targets on prefetching-friendly or dalay-tolerant data in the managed apps, while the real-time part won't be affected. For example, in SNS apps, if a user wants to share a picture with her friends, it will be uploaded to SNS server immediately regardless the bandwidth condition. If her friend refreshes the SNS app before AppATP prefetches the picture, it will be downloaded immediately. The real-time transmissions behave the same with the ordinary SNS apps without hurting user experience. On the other hand, her friends may not immediately check (download) it. There could be a time lag before her friends open the SNS app and read (download) it. In this case, we regard the picture as a prefetching-friendly data, which can be potentially prefetched in good connectivity during the time lag. Even if the prefetching fails, it just degenerates to the normal circumstance: downloading the picture immediately when her friends open and refresh the SNS app.

As illustrated in Fig. 1, we propose AppATP, an application-layer protocol to intelligently manage mobile-cloud data transfers. It employs the cloud as a relay between mobile devices and content providers for data transmission. Having obtained the authorization to access data in prefetch-friendly and delay-tolerant apps on mobile devices, AppATP will run lightweight *duplicates* of these apps on the cloud side, request data from original content providers periodically and store them in the cloud to wait for transmission. For clarity, in the following, we use the downlink transmission as an illustration, where a mobile device downloads from the cloud, since energy consumption of 3G or WiFi interfaces is much more for downlink than for uplink in most popular apps [7]. However, it is easy to generalize the protocol design to the case involving both downlinks and uplinks, which will be discussed in Sec. 5. In particular, AppATP has the following unique advantages:

Cloud-based Multi-App Coordination: As apps are usually independent of each other and rely on different content providers (e.g., Netflix, iCloud, Facebook) deployed on different hosts (e.g., Amazon EC2, Microsoft Azure, private servers), it is challenging to jointly gather and manage data for multiple apps from various content servers. Rather than implementing a stand-alone protocol in each app on the mobile end, AppATP gathers and stores the data for different apps in the cloud and schedules data transfers using an *unified* algorithm, as shown in Fig. 2. Since a cloud platform has abundant computing resources and is well connected to multiple carriers and ISPs using high-speed links, the cloud-based AppATP serves as a powerful agent between mobile devices and content providers. Furthermore, as we believe that more and more mobile apps will be cloud-based in a long

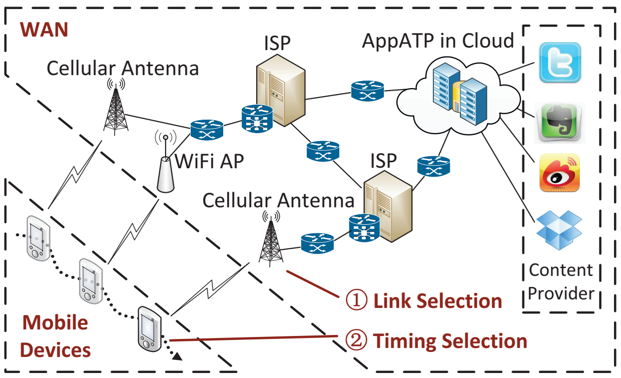


Fig. 2: Using cloud to provide data management for mobile devices.

run, designing and implementing AppATP in the cloud platform would make it more applicable and extensible for abundant apps in the future [9].

Online Scheduling: Due to the stochastic characteristics of wireless network conditions and the mobility of users, it is difficult, if not impossible, to accurately predict the available bandwidth in the future. Even if the network conditions are given, it is still challenging to tell when is the proper time for transmission. To cope with the intrinsic stochastic nature of wireless networks, we apply *Lyapunov optimization framework* [10] to design an *online* transmission control protocol, which only relies on the current bandwidth information and data queue backlogs to make scheduling decisions, and does not need any prediction.

Tuning the Energy-Delay Tradeoff: Scheduling may incur delays, if transmission is deferred to a later time. The choices in energy-delay tradeoff may vary with application types and user contexts. Users may require shorter delays when the energy is ample or the response time is stringent, while preferring energy conservation when the battery has low power. AppATP is able to quantitatively tune the energy-delay tradeoff in the scheduling. By adjusting a single parameter in our control algorithm, AppATP can adaptively balance such a tradeoff according to application requirements and user preferences.

Computation Offloading: To conserve transmission energy without incurring any energy overhead in terms of computation and networking, AppATP offloads most of its workload to the cloud, causing the minimum overhead on resource-constrained mobile devices. First, the computation-sensitive work is done in the cloud side, i.e., the maintenance of multiple queue backlogs, data scheduling, and online transmission decisions. Second, as the cloud end works as an agent between mobile end and its various content providers, AppATP only needs to probe the connectivity between mobile end and cloud end once every cycle phase, rather than probing the connectivity to each content provider. This mitigates the energy overhead of bandwidth estimation without losing accuracy.

Note that an indispensable part in AppATP design is to select the best wireless link among candidate links, e.g., 2G/3G/4G, WiFi access points (APs). We assume that the mobile device will always select a preferable link from all the available ones, by using existing link selection mechanisms [11]. As shown in Fig. 2, we focus on designing a connectivity-aware protocol to adaptively find the proper time for data transfers, while link selection is not our focus.

3 System Models

We consider a mobile user who is running N different apps $\mathcal{N} \triangleq \{0, 1, 2, \ldots, N\}$, and thus N duplicates on the cloud, as shown in Fig. 1. The data (content) for each app arrives at the corresponding queue on the cloud side, to be downloaded to the mobile device in proper time slots $t = 0, 1, 2, \ldots$, assuming a discrete time-slotted model (which is easy to realize with the controller on the cloud). Denote $\mathbf{Q}(t) \triangleq (Q_1(t), \ldots, Q_N(t))$, where $Q_i(t)$ represents the queue backlog of data to be transmitted from the cloud to app i on the mobile device at the beginning of time slot t. In each time slot t, we denote the amount of new data that arrive for each app i (arrive to its corresponding queue on the cloud) as a random variable $A_i(t)$, with $\mathbf{A}(t) \triangleq (A_1(t), \ldots, A_N(t))$ denoting the vector of data arrival rates. We assume that each random variable $A_i(t)$ is i.i.d. over time slots t, with their expectations given by $\mathbb{E}\{\mathbf{A}(t)\} = \boldsymbol{\lambda} \triangleq (\lambda_1, \ldots, \lambda_N)$. Furthermore, since the workload in a mobile app is highly dynamic and usually unpredictable (e.g., the friends in SNS may share content at random time), we do *not* assume any *a priori* knowledge of the statistics of $A_i(t)$ for any app i.

3.1 Bandwidth and Decision Models

As recent studies indicate that bandwidth is the critical factor that affects energy consumption of wireless data transmission, we use $\omega(t)$ to denote the connectivity condition in time slot t, in terms of the achievable downlink bandwidth of the currently used wireless AP. The connectivity condition $\omega(t)$ varies randomly in two aspects:

First, a typical smartphone can usually access multiple links such as a persistent 3G interface and several WiFi APs, but can only activate one of them at a time. The actual connectivity depends on the quality of the selected link. We have $\omega(t) \in \{B_{\text{3G}}(t), B_{\text{WiFi}}(t)\}$, where $B_{\text{3G}}(t)$ denotes the 3G bandwidth, and $B_{\text{WiFi}}(t)$ denotes the WiFi bandwidth. Clearly, $\omega(t)$ equals to $B_{\text{3G}}(t)$ (or $B_{\text{WiFi}}(t)$) if the device chooses the 3G (or WiFi) interface at time t. The issue of choosing 3G or WiFi and choosing the best WiFi APs from all available ones for energy saving have been discussed in previous works [11]–[13]. In this paper, we assume that a preferable AP has been selected using a simple and widely used link selection algorithm proposed in [11], in which the mobile device selects the link with the best connection by running a series of probe-based tests to the cloud.

Second, even after a particular link is selected, the connectivity still keeps changing due to user mobility, flash crowds in the chosen link, the limited coverage of WiFi APs, or even under a bad weather, etc. Given a selected link, our design of AppATP focuses on deciding the proper timing (time slots of higher bandwidth) to transfer data over this link. Given the current achievable bandwidth $\omega(t)$ of the mobile device and queue backlogs $\mathbf{Q}(t)$ maintained by the cloud, AppATP needs to make a decision

$$\alpha(t) \in \Omega \triangleq \{\text{“Transmit } Q_i(t)\text{”}, \text{“Idle”}\},$$

i.e., whether to download data from the cloud to the mobile device in time slot t, and if yes, the data of which app i to download. Hence $\alpha(t) =$ “Transmit $Q_i(t)$” if AppATP decides to transmit data queued in $Q_i(t)$ to app i, and $\alpha(t) =$ “Idle” if all data transfers are deferred for energy saving in time t.

For each app i, let $b_i(t)$ denote the amount of data transmitted from the cloud to the mobile device in time slot t. Note that $b_i(t)$ is a function of the current downlink bandwidth $\omega(t)$ and transmission decision $\alpha(t)$:

$$b_i(t) = \begin{cases} \omega(t)\tau, & \text{if } \alpha(t) = \text{“Transmit } Q_i(t)\text{”}, \\ 0, & \text{if } \alpha(t) = \text{“Idle”}, \end{cases} \tag{1}$$

where τ represents the time span of one time slot. In our transmission algorithm, we need to estimate the achievable 3G or WiFi bandwidth $\omega(t)$ at the beginning of each time slot t, which may incur high overhead if the length τ of a time slot is too small. However, if τ is too large, it is inappropriate to let $\omega(t)$ represent the network condition over the entire time slot t since the bandwidth could vary considerably. According to the observed bandwidth statistics in our traces, we empirically choose a moderate value of $\tau = 60$ seconds. For model tractability, we assume that the bandwidth remains unchanged during a time slot, which is reasonable when τ is relatively small [12], [14]. Over a long time period T, let M denote the total amount of data transmitted by AppATP. Clearly, we have $M = \sum_{t=1}^{T} \sum_{i=1}^{N} b_i(t)$.

3.2 Energy Consumption Models

We denote the energy consumed by data transmissions on a mobile device in time slot t as $P(t)$, which depends on the current downlink bandwidth $\omega(t)$ and transmission decision $\alpha(t)$, i.e., $P(t) = P(\omega(t), \alpha(t))$. Over a time period T, the total energy consumption on the mobile device is $E_{\text{AppATP}} \triangleq \sum_{t=1}^{T} P(t)$. In the following, we will describe the detailed model for $P(t)$ adopted by AppATP. AppATP employs practically measured energy consumption models of 3G and WiFi interfaces on modern smartphones [6], [15], which are commonly used in the latest solutions to mobile energy accounting and management [14], [16]. Note that according to [6], [15], both uploading transmission and downloading transmission in mobile devices can share the same energy model as below.

Energy Model for 3G: The energy consumption of data transmission over 3G networks mainly consists of two parts: *transmission energy* and *tail energy*. The *transmission energy* is proportional to the length of transmission time and transmission power level [15]. After transmitting a packet, instead of transiting from the high to low power state immediately, the 3G interface spends substantial time in the high power state to avoid the delay and signalling overhead that would otherwise be incurred by frequent power state transitions, channel allocations and releases. Such a mechanism causes the dissipation of the *tail energy* [16], which can not be ignored in energy management. Since lingering in the high power state consumes more energy, a fixed tail time T_{tail} is empirically used to set the inactivity timer according to the performance-energy tradeoff [15]. Let t_{intvl} denote the time interval between the current transmission and the last transmission, then we have $t_{\text{intvl}} \in (0, T_{\text{tail}})$.

In summary, the energy consumption for transmitting data via 3G in time slot t can be expressed as $P(t) = P_{\text{3G}}\tau + P_{\text{tail}}t_{\text{intvl}}$, where P_{3G} and P_{tail} represent the transmitting power coefficient and tail power coefficient, respectively. More details of the transitions between different 3G power states can be found in [15], which will be incorporated into our calculations in Sec. 6.

Energy Model for WiFi: The energy consumption for data transmission via WiFi networks also consists of two parts. *First*, an *initial cost* is incurred by scanning and associating with an AP [6]. As we focus on the energy saving during data transmissions, we assume that a preferable AP has already been selected based on the link selection scheme mentioned in Sec. 3-A. The incurred initial cost is excluded in our model. Once associated with an AP, the WiFi interface on smartphones typically follows the Power Save Mode (PSM) [15], under which the energy needed to keep it on is small. *Second*, the *transmission energy* of a WiFi connection is proportional to the length of transmission time and the transmission power level [6]. As a result, the energy consumed for transmitting data via WiFi in time slot t can be expressed as $P(t) = P_{\text{WiFi}}\tau$, where P_{WiFi} is the transmitting power coefficient of WiFi.

To summarize, integrating the above detailed energy models for 3G and WiFi, the mobile device energy consumption under the transmission decision $\alpha(t)$ made by AppATP in time slot t is given by

$$P(t) = \begin{cases} P_{\text{3G}}\tau + P_{\text{tail}}t_{\text{intvl}}, & \text{if } \alpha(t) \neq \text{Idle},\ \omega(t) = B_{\text{3G}}(t), \\ P_{\text{WiFi}}\tau, & \text{if } \alpha(t) \neq \text{Idle},\ \omega(t) = B_{\text{WiFi}}(t), \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

4 Energy-Efficient Transmission Control

In this section, we present our algorithm for transmission energy conservation under bounded delays. According to Little's law in queueing theory, the delay translates into the queue backlog in a stable system. Therefore, AppATP aims to minimize the transmission energy over the long run under bounded queue backlogs.

We define the *time-averaged energy consumption* of the mobile device as

$$\overline{P} \triangleq \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{|P(t)|\}. \tag{3}$$

To minimize the energy consumption, a baseline strategy is to transmit data only when the network connectivity (downlink bandwidth) is good enough. Nonetheless, if we aggressively defer data transmissions for power conservation, queue backlogs $\mathbf{Q}(t)$ of all the apps will increase unboundedly, leading to unacceptable delays and poor user experience. Hence, an *energy-delay tradeoff* must be considered when scheduling data transmissions.

To strike a balance in such a tradeoff, we require all the queues to be stable in the time average sense, i.e.,

$$\overline{Q} \triangleq \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i=1}^{N} \mathbb{E}\{|Q_i(t)|\} < \infty, \tag{4}$$

where $\overline{Q}$ represents the *time-averaged queue backlog* and the queueing dynamics can be characterized by

$$Q_i(t+1) = \max[Q_i(t) - b_i(t), 0] + A_i(t). \tag{5}$$

If the condition in Equation (4) is satisfied, then all the data that have arrived to the queues in the cloud will be transmitted to the mobile device in bounded time. Moreover, based on Little's law, a larger $\overline{Q}$ implies longer delays for the apps.

In each time slot t, AppATP makes an online transmission decision $\alpha(t) \in \Omega$, with the objective of minimizing the time-averaged energy consumption under finite queue sizes for all apps, leading to the following stochastic optimization problem:

$$\text{minimize}_{\alpha(t)\in\Omega} \quad \overline{P} \triangleq \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{|P(t)|\} \tag{6}$$

$$\text{subject to} \quad \overline{Q} < \infty. \tag{7}$$

4.1 An Online Stochastic Control Algorithm

Due to the varying conditions of wireless networks and the mobility of users, it is difficult, if not impossible, to accurately predict downlink bandwidth in the future. Hence, in AppATP, we leverage the current downlink bandwidth information to make transmission decisions. The requirement above motivates us to optimize the time-averaged energy consumption using Lyapunov optimization [17], [18], which has been successfully used in energy optimization problems [12], [19]–[21] in other applications. Leveraging this framework, we develop a lightweight algorithm that only utilizes the current state (i.e., network bandwidth $\omega(t)$, queue backlogs $\mathbf{Q}(t)$ and data arrival rates $\mathbf{A}(t)$) to solve our specific problem (6).

We first define a Lyapunov function, $L(\mathbf{Q}(t))$, which is a scalar metric of queue congestion [10] that reflects the delays of app data to be transmitted, as follows:

$$L(\mathbf{Q}(t)) \triangleq \frac{1}{2}\sum_{i=1}^{N} Q_i^2(t). \tag{8}$$

It is clear that $L(\mathbf{Q}(t)) \geq 0, \forall t$. A smaller value of $L(\mathbf{Q}(t))$ implies that all queue backlogs are small in the average sense, while a larger value of $L(\mathbf{Q}(t))$ implies that at least one queue (for one app) has large backlog. To keep queue stability by persistently pushing the Lyapunov function towards a less congested state, we introduce the Lyapunov drift $\Delta(\mathbf{Q}(t))$:

$$\Delta(\mathbf{Q}(t)) \triangleq \mathbb{E}\{L(\mathbf{Q}(t+1)) - L(\mathbf{Q}(t))|\mathbf{Q}(t)\}, \tag{9}$$

which is the expected change in the Lyapunov function over one time slot, given that the current state in time slot t is $\mathbf{Q}(t)$. We now incorporate the expected energy consumption over one time slot to both sides of (9), which leads to a *drift-plus-penalty* term: $\Delta(\mathbf{Q}(t)) + V\mathbb{E}\{P(t)|\mathbf{Q}(t)\}$. With Lyapunov optimization, the objective of optimal control decisions on energy minimization and queue stability can be achieved by minimizing the drift-plus-penalty term in every slot (pp. 39, [10]).

The control parameter $V > 0$ represents a *design knob of the energy-delay tradeoff*, i.e., how much we shall emphasize the energy-minimization (6) compared to the transmission delay (7). It empowers AppATP to make flexible design choices between transmission delay and energy consumption according to the application type and user context. For example, when the mobile device is running out of power, the user may allow a longer delay of data synchronization in Dropbox for energy conservation. AppATP can handle this situation by setting a larger value of V.

Rather than directly minimizing the drift-plus-penalty term $\Delta(\mathbf{Q}(t))+V\mathbb{E}\{P(t)|\mathbf{Q}(t)\}$ in each slot, the *min-drift-plus-penalty* algorithm in Lyapunov optimization [10] seeks to minimize an upper bound of it. We derived an upper bound on the drift-plus-penalty term in our specific problem and it is stated in the following lemma:

***Lemma** 1:* For $V > 0$, given any possible wireless network condition $\omega(t)$, data queue backlogs of all the apps $\mathbf{Q}(t)$, and data arrival rates $\mathbf{A}(t)$, for all possible action $\alpha(t)$, we have

$$\Delta(\mathbf{Q}(t)) + V\mathbb{E}\{P(t)|\mathbf{Q}(t)\} \leq B + V\mathbb{E}\{P(t)|\mathbf{Q}(t)\} + \sum_{i=1}^{N}\mathbb{E}\{Q_i(t)(A_i(t) - b_i(t))|\mathbf{Q}(t)\}, \tag{10}$$

where $B = (A_{max}^2 + b_{max}^2)/2$, and $A_{max} \geq A_i$, $\forall i \in \mathcal{N}$, represents the maximum amount of data that can arrive for any app i per time slot and $b_{max} \geq b_i$, $\forall i \in \mathcal{N}$, represents the maximum amount of data that can be transmitted via the wireless network in a time slot.

Proof: Please refer to the Appendix for the proof. ■

Based on Lemma 1, we next attempt to minimize the upper bound of the drift-plus-penalty, namely the term on the RHS of (10), in each time slot t. Interestingly, given $\mathbf{Q}(t)$ and $\omega(t)$, the control decision $\alpha(t)$ can only affect the energy consumption $P(t)$ and the amount of data $b_i(t)$ transmitted in a time slot t. Leveraging the concept of opportunistically minimizing an expectation (pp. 13, [10]), we seek to minimize the following simplified term, which eventually minimizes the RHS of (10):

$$VP(t) - \sum_{i=1}^{N} Q_i(t)b_i(t). \tag{11}$$

Algorithm 1: *Energy-efficient transmission strategy*

```
1: At the beginning of each time slot $t$, monitor the queue backlog $Q_i(t)$ of each app $i$ on the cloud side and estimate the current bandwidth $\omega(t)$ on the mobile device.
2: Minimize (11) to yield the control decision $\alpha(t)$, where $P(t)$ and $b_i(t)$, as functions of $\alpha(t)$ and $\omega(t)$, are given by (2) and (1), respectively.
3: If $\alpha(t)$ = "Transmit $Q_i(t)$", download the data in app $i$ from the cloud to the mobile device.
   If $\alpha(t)$ = "Idle", stay idle for energy conservation.
4: At the end of time slot $t$, update the queue backlog $Q_i(t+1)$ of each app $i$ on the cloud side using (5).
```

Specifically, our algorithm is described in Algorithm 1, which makes a decision $\alpha(t)$ drawn from the set $\Omega \triangleq \{$"Transmit $Q_i(t)$", "Idle"$\}$ in each time slot t. Note that Algorithm 1 has a low complexity of $O(N)$, since minimizing (11) only requires searching for the minimum value in a one-dimensional array of length N.

Remarks: Algorithm 1 has the following phyisical implications: Considering a fixed V, if the control action $\alpha(t)$ = "Idle", we choose not to transmit any data in time slot t, then we have $P(t) = 0$ and $b_i(t) = 0$, $\forall i \in \mathcal{N}$. Accordingly, (11) equals to zero. As we prefer the control decision which minimizes (11), the transmission will take place only if there exists an $\alpha(t)$ that drives (11) to be less than zero. This can happen when either the network bandwidth $\omega(t)$ is high, and thus more data $b_i(t)$ can be transferred, or when a certain queue $Q_i(t)$ is already congested in time slot t. In other words, as illustrated in Fig. 3, AppATP tends to transmit data under the following conditions: 1) when the network connectivity is good enough for the mobile device to download more data with less energy consumption; or 2) when the queues of certain apps are congested, so that data transmission must be triggered to maintain queue stability.

4.2 Performance Analysis

Due to the randomness of wireless network condition and data arrival rates, Algorithm 1 could not give an exact solution to problem (6). However, the time-averaged energy consumption and queue backlog are guaranteed

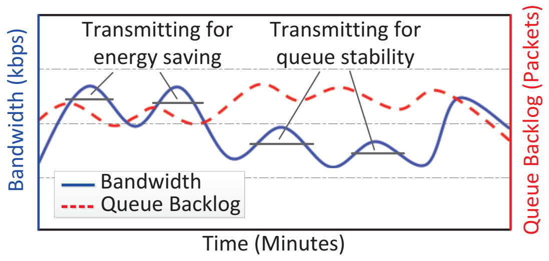


Fig. 3: The data transmission takes place when network bandwidth is high or one of queue backlogs is large.

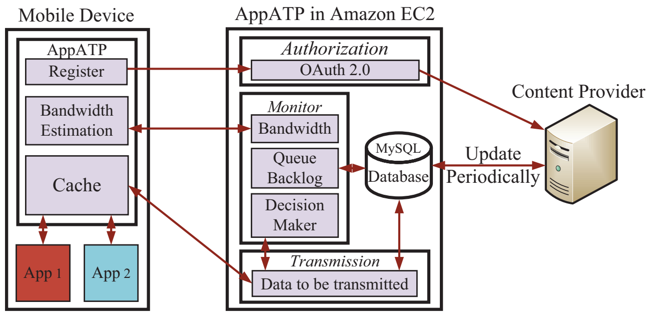


Fig. 4: The AppATP framework developed on Amazon EC2.

to be optimized down to definite bounds (following a methodology in pp. 47, [10]).

Let P^* denote the minimum time-averaged energy consumption $\overline{P}$ that can be achieved by any control policy that satisfies queue stability. Let Λ be the set of all arrival rates that the mobile device can stably support [22], such that the total data arrivals for all the apps in a time slot will not exceed $B_{max}\tau$, which is the maximum amount of data that can be transmitted in a time slot under the maximum network bandwidth. We can bound the performance of Algorithm 1 in our specific problem as follows:

***Proposition** 1:* (*Performance Bounds*) Assume that the data arrival rate vector $\boldsymbol{\lambda}$ is strictly within the network capacity region Λ defined above, then Algorithm 1 achieves the following performance bounds of the time-averaged energy consumption and queue backlog:

$$\overline{P} = \limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\{|P(t)|\} \le P^* + \frac{B}{V}, \tag{12}$$

$$\overline{Q} = \limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{N} \mathbb{E}\{|Q_i(t)|\} \le \frac{B+VP^*}{\epsilon}, \tag{13}$$

where $\epsilon > 0$ represents a measure of the distance between the data arrival rate $\boldsymbol{\lambda}$ and the network capacity region Λ.

Proof: Please refer to the Appendix for the proof. ■

Remarks: Note that (12) and (13) characterize the energy-delay tradeoff within $[O(1/V), O(V)]$. Specifically, we can use an arbitrarily large value of V to drive the time-averaged energy consumption $\overline{P}$ arbitrarily close to the optimal P^* at a cost, as (13) implies that the time-averaged queue backlog $\overline{Q}$ grows linearly with V. Such an energy-delay tradeoff allows AppATP to make flexible design choices and find the value of V such that further increasing V yields very small reduction in $\overline{P}$. We will discuss the selection of V through practical experiments in Sec. 6.

5 Prototype Implementation

As illustrated in Fig. 4, AppATP is composed of a mobile-end middleware and a cloud-end data manager. On the cloud side, we have developd these components in Java and deployed them on Amazon EC2. In the mobile device, AppATP consists of a bandwidth estimator and a cache. We have developed the components (in Java) for mobile running Android Jelly Bean.

5.1 The Cloud-end Components of AppATP

The AppATP data manager at the cloud end has two main functions: (1) gather data from different original app servers, (2) make the transmission decision according to app queue backlogs and the current bandwidth estimation. The framework of AppATP manager is presented in the right part of Fig. 4, which consists of three services: *authorization*, *monitoring* and *transmission*.

Authorization: To access data from different apps on the cloud side, AppATP adopts the OAuth 2.0 [23] open protocol to allow secure authorization via standard method from web, mobile and desktop applications. OAuth 2.0 grants AppATP limited access to application data in the original content servers (e.g., the latest updates in one's Facebook and Twitter accounts) via HTTP requests. If an app i (e.g., News Reader, File Downloader) has potential energy benefits from prefetching or deferring transmissions, the register component in the mobile-end will register the app in the AppATP Authorization Service within the cloud, which then guides the user to the authorization webpage of the original content provider. The user needs to input her ID and password to confirm this authorization. After successful authorization, AppATP will periodically query the content providers and update the data queue backlog $Q_i(t)$ for each registered app i, e.g., pull the latest shared content from Sina Weibo servers to the cloud-side storage every 5 minutes. Data from different apps are labeled with distinct keys (i.e., app ID, timestamp, byte length, time to live, etc.) and stored in a MySQL database.

Monitoring and Transmission: The queue backlogs of all the apps are maintained by AppATP Monitor Service. When new data arise in an app, the Monitor Service will update the corresponding queue size. The Monitor Service also receives bandwidth estimation $\omega(t)$ via HTTP messages from mobile devices periodically (at the beginning of every time slot t). The detailed mechanisms for estimating the bandwidth will be discussed in the following Sec. 5.2. Based on the bandwidth estimation $\omega(t)$ and the app queue backlogs $\mathbf{Q}(t)$, the Monitor Service makes transmission decisions according to Algorithm 1. If the transmission decision is not idle, it will send a notification with the app queue ID i to be transmitted. After receiving the transmission notification from the Monitor Service, AppATP Transmission Service will pull data from the head of the chosen queue in the database, pack them into JSON (JavaScript Object Notation) format, and load them to a TCP socket for the mobile device to download.

Weighted Queue Management: For certain delay-tolerant apps such as file/video downloading or app updates, there will be requests for bulk data. Such bulk data make their queue backlogs quite large at the very beginning. Since Algorithm 1 will choose to transmit data in long queues so to guarantee queue stability (4), the bulk data will be continuously transmitted without considering energy efficiency. Such bulk data also incur longer delays for other apps, as Algorithm 1 transmits data in the longest queue when bandwidth is sufficient. To address this practical problem in the implementation, AppATP assigns a weight for each queue. The transmission priority of each app can also be tuned by this weight. In particular, for bulk data downloading, we set a dynamic weight to make the corresponding queue size always equal to the median length of all the queues in the system. Thus, the data can be downloaded at a proper time, without affecting other apps.

5.2 The Mobile-end Components of AppATP

The AppATP middleware at the mobile end has two main functions: (1) estimate the bandwidth between the cloud and the mobile device, (2) cache the data transmitted from the AppATP manager in cloud and distribute the data cached to the corresponding app when it is called. The framework of AppATP app in the mobile-side is presented in the left part of Fig. 4, which consists of two components: *bandwidth estimator* and *cache*.

Adaptive Bandwidth Probing: According to Algorithm 1, we need to estimate the current bandwidth $\omega(t)$ to make the transmission decision in every slot. There are two methods to estimate wireless bandwidth: *probe-based* [11] and *signal-strength-based* [12] approaches. Estimates based on received signal strength indicator (RSSI) are coarse since wireless network bandwidth only partially depends on RSSI. Moreover, even though RSSI-based approach incurs little overhead (since one can use Android RSSI API to estimate bandwidth), the tail energy overhead, as described in Sec. 3.2, is inevitable when reporting results to the cloud via 3G. On the other hand, probe-based approaches are more accurate, yet relying on transmitting data probes, which incurs higher energy overhead. Although we may use the packet in the longest queue backlog as the probe to reduce the energy of transmitting irrelevant data, the subsequent tail energy still exists.

Compared to conventional Lyapunov based algorithm [5] that needs frequent bandwidth probing every time slot, which costs significant tail energy wastage, AppATP employs a practical approach to reduce bandwidth estimation overhead. Inspired by the congestion control protocol in TCP, we adopt an adjustable probing interval for bandwidth estimation. Initially, the interval between the current and last estimation is set to one time slot. If the current estimated bandwidth is close to the last estimate, the time interval will be increased by one slot. However, if the bandwidth changes by more than 50% as compared to the last probe, we shorten the interval by one half. This adaptive algorithm is described in Algorithm 2. Different from bandwidth estimation in Ethernet networking (e.g., data centers) [24], bandwidth of wireless networks largely depends on the location of the user and a user's mobility is usually confined within a certain period, this heuristic can effectively avoid unnecessary probes, while guaranteeing estimation accuracy. The effectiveness and accuracy of Hence, this adaptive bandwidth probing approach will be verified in Sec. 6.

Algorithm 2: *Adaptive Bandwidth Estimation*

```
Require: The slot interval between the latest 2
    bandwidth measurement $T_{\mathsf{intvl}} = 1$. The last
    (current) measured bandwidth $B_{\mathsf{L}} = 0$ ($B_{\mathsf{C}} = 0$).
    The difference threshold between the latest two
    measured bandwidth $\kappa = 0.3$. The slot count from
    last bandwidth measurement $i = 1$.
 1: for slot $t = 0, 1, 2, \dots$ do
 2:   if $i == T_{\mathsf{intvl}}$ then
 3:     $i = 0$
 4:     $B_{\mathsf{L}} = B_{\mathsf{C}}$
 5:     $B_{\mathsf{C}}$ =Probe-based bandwidth measurement
 6:     $\omega(t) = B_{\mathsf{C}}$
 7:     if $|B_{\mathsf{C}} - B_{\mathsf{L}}| < B_{\mathsf{L}} \times \kappa$ then
 8:       $T_{\mathsf{intvl}} + +$
 9:     else
10:       $T_{\mathsf{intvl}} = T_{\mathsf{intvl}}/2$
11:     end if
12:   end if
13:   $i + +$; $\omega(t) = B_{\mathsf{C}}$
14: end for
```

Smart Cache Management: On the mobile device, all the packets transmitted via AppATP are stored in the same cache file in the SD card. We use the app ID of each packet to identify the app to which it belongs. When users open a registered app and request data, the app will use its unique ID to draw the corresponding packets from the cache.

For delay-tolerant apps, the data deferred by AppATP have a high probability to be consumed in the future. For example, delay-tolerant data transmitted for software update will be consumed sooner or later. However, for prefetch-friendly apps, prefetching techniques typically have the issues of cache size and hit rates. As mobile devices are resource-constrained, our cache file is allowed a maximum size of 200 MB. When the volume of data is large, some prefetched data might not be consumed yet before getting overwritten by incoming data. To avoid such wastage, AppATP simply stops transmitting data when the cache is full. Futhermore, we allow a bigger cache size cap for those apps that are frequently used and incur more traffic. On the other hand, it is possible that the user does not use an app for a long time. We thus introduce a timer for each registered app. If an app has not been used for a fixed period, AppATP will stop transmitting data for this app and remove corresponding data in the cache until it is active again, at which point the timer is reset.

When the data prefetched by AppATP is not used by the user, the energy consumption in prefetching is wasted. To reduce such energy cost in cache misses, we can utilize cloud resources to employ machine learning techniques to customize prefetching strategies based on user behaviors, which can increase the cache hit rate. This is an interesting open future work. In case that the cached data is outdated, before returning data to apps, AppATP sends the timestamp of the latest prefetched packet to the content server to check whether there is new update. If the latest data has not been cached yet, AppATP will immediately download the data and return it to the corresponding app.

To incorporate *uplink* transmission management in AppATP, we store the data to be uploaded in the cache as uplink queues. The mobile device estimates the uplink bandwidth and reports it with the uplink queue backlogs to the cloud periodically. The Monitor Service then maintains the information and makes uplink transmission decisions.

6 Performance Evaluation

In the performance evaluation, we suppose that mobile users may run apps and request data in random time slot. We compare AppATP with the benchmark scheme in which no data is intentionally prefetched or delayed: the data are transmitted instantly to the mobile device. We evaluate AppATP and its parameter choices through both trace-driven simulations and experiments based on our prototype implementation.

6.1 Trace-Driven Simulations

Trace Collection and Parameter Settings: To evaluate how AppATP performs under realistic network conditions, we collect wireless bandwidth traces under representative scenarios. Specifically, during an 18-hour period (8:00 - 24:00), we took a bus and traveled around downtown, and then walked around in a campus, carrying an Android smartphone equipped with WiFi and 3G interfaces. The phone could only use one interface at a time, and prioritizes WiFi if available. Every minute, we measured the downlink bandwidth by downloading 1 MB of data from the cloud. The traces show that the median 3G downlink bandwidth is $1,852$ kbps. WiFi is available 18.8% of the time and the median WiFi downlink bandwidth is 964 kbps, with a median session length of 35.4 minutes.

To simulate apps with different volumes of data, we determine data arrival rates according to the data patterns and traffic volumes of emerging apps on mobile phones [1]. We consider 10 running apps and assume that their data arrivals all follow *Poisson Processes* with a mean arrival rate $\mathbb{E}\{A_i(t)\} = \lambda_i = \lambda$ for all i, where λ is chosen from $\{1.0, 2.0, 3.0, 4.0\}$ packets/minute. The size of one packet is 100 KB. Every 60 seconds, we use the downlink bandwidth in our traces as the bandwidth estimate $\omega(t)$ in time slot t, and run AppATP based on $\omega(t)$ and queue backlogs $\mathbf{Q}(t)$. The energy overhead of bandwidth estimation is also included in the simulation. According to the power models developed in [6], [15], we set the power coefficients P_{3G}, P_{tail}, and P_{WiFi} to 1.5 W, 0.7 W and 1.0 W, respectively.

We compare the total energy consumption $E_{\mathsf{AppATP}} = \sum_{t=0}^{T-1} P(t)$ with that of the benchmark scheme without AppATP $E_{\mathsf{w/o}}$. To calculate $E_{\mathsf{w/o}}$, we use the average network condition observed in the past as a reasonable approximation:

$$E_{\mathsf{w/o}} = \frac{M}{\overline{B_{\mathsf{WiFi}}}} \cdot P_{\mathsf{WiFi}} \cdot A_{\mathsf{WiFi}} + \left(\frac{M}{\overline{B_{\mathsf{3G}}}} \cdot P_{\mathsf{3G}} + T_{\mathsf{tail}} \cdot P_{\mathsf{tail}}\right) \cdot (1 - A_{\mathsf{WiFi}}),$$

where $\overline{B_{\mathsf{3G}}}$ and $\overline{B_{\mathsf{WiFi}}}$ represent the average bandwidth of 3G and WiFi, respectively, $M = \sum_{t=0}^{T-1} \sum_{i=1}^{N} b_i(t)$ represents the total amount of data transmitted by AppATP, and A_{WiFi} represents the average observed availability of WiFi.

Simulation Results: In our simulations, the test duration is $1,000$ time slots for each parameter setting. We set up three groups of apps in the cloud, with arrival rates $\lambda = 2.0, 3.0, 4.0$. We run each group of apps separately with both a fixed V and an adaptive V, where V is 20 times the data arrival rate, and without AppATP. The same amount (20 MB) of data is transmitted for each group. We compare the energy consumptions of the three schemes. As shown in Fig. 5, both AppATP with a fixed V and an adaptive V can signficantly save energy as compared to the naive random strategy. In particular, given a fixed V, the energy consumption increases from 43 J to 60 J as the arrival rate λ grows from 2.0 to 4.0, indicating a decrease of the energy saving from 26% to 11%. However, an adaptive V can consistently save energy by 26%. The tradeoff between the energy saving and transmission delay can be balanced via the parameter V. Fig. 5 implies that a better strategy is to adaptively choose V based on the network condition and average data arrival rates.

We also compare the average throughput achieved with and without AppATP, as V varies. The average throughput is calculated as the total transmitted data divided by the total effective transmission time. As shown in Fig. 6, AppATP achieves a higher throughput as it utilizes higher bandwidth for transmission. Especially, the throughput increases fast when $V < 70$ and grows slowly if $V > 70$. The reason is that when we set a larger V to emphasize on energy conservation, AppATP tends to transmit only when bandwidth is ample. Nonetheless,

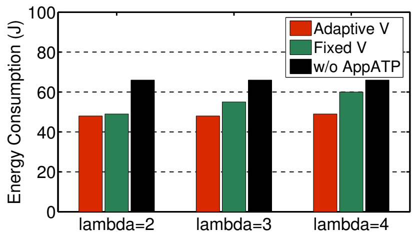


Fig. 5: Energy consumption vs. the data arrival rate under different strategies.

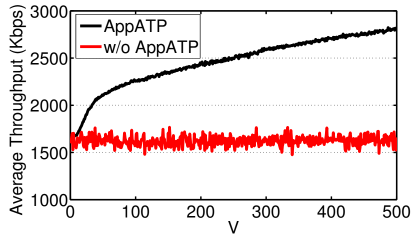


Fig. 6: The impact of V on the average transmission rate.

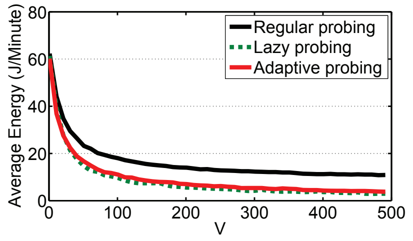


Fig. 7: The time-avg energy consumption $\overline{P}$ under different bandwidth estimation strategies.

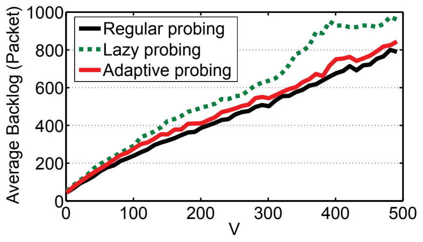


Fig. 8: The time-avg queue backlogs $\overline{U}$ under different bandwidth estimation strategies.

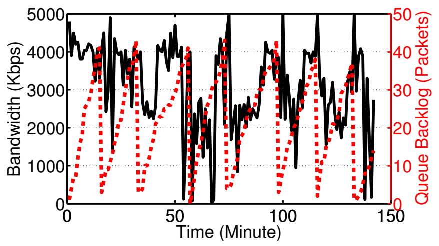


Fig. 9: The bandwidth and total queue backlog over time in a test run of a single app.

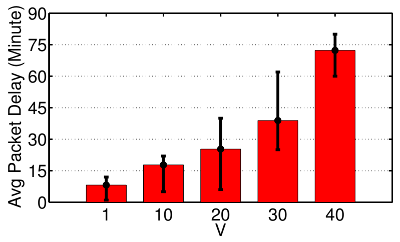


Fig. 10: The impact of V on the transmission delay in real-world measurement.

even with an aggressive V, to guarantee queue stability, AppATP needs to transmit at times under suboptimal bandwidth, which incurs a slower growth when $V > 70$.

We next evaluate the impact of V on energy consumption and delay. As bandwidth estimation incurs energy overhead, we run AppATP under three bandwidth estimation schemes: regular probing [5] (probing bandwidth every minute), lazy probing (probing bandwidth every 20 minutes), and adaptive probing (as described in Sec. 5), and compare their performance. *First*, regardless of the bandwidth estimation scheme used, given the arrival rate $\lambda = 1.0$, the time-averaged energy consumption drops quickly at first and then slowly, as shown in Fig. 7, while the time-averaged queue backlog grows linearly as shown in Fig. 8. This finding confirms the $[O(1/V), O(V)]$ energy-delay tradeoff as captured in (12) and (13). Particularly, there exists a *sweet spot* of V (e.g., $V = 100$ when $\lambda = 1.0$), beyond which increasing V leads to marginal energy conservation yet consistently growing queue sizes, which means higher delays by Little's Theorem.

We further compare the energy overhead and delay performance of different bandwidth estimation schemes. As we can see in Fig. 7, the energy consumptions of lazy probing and adaptive probing are obviously lower than that of regular probing, with a gap around 10 J. The reason is that they incur different energy overhead: the averaged times of bandwidth estimation in the three algorithms are 50, 163.22, and 1000, respectively. The fewer times of estimation, the less tail energy wastage in mobile devices. On the other hand, Fig. 8 shows that the averaged queue backlog of lazy probing is much higher than regular probing and adaptive probing, since estimating bandwidth in a fixed long period can not reflect the real-time bandwidth, and prevents AppATP from choosing the good timing for data transmissions. Adaptive probing can reduce bandwidth probing energy overhead as well as capture the real-time bandwidth variation, simultaneously achieving the advantages of both frequent and lazy probing.

6.2 Real-World Experiments

Experimental Setup: To test AppATP on prefetch-friendly apps, we developed a mixed social networking app, *MixSNS*, in about $5,000$ lines of Java code. We let *MixSNS* employ AppATP to prefetch all the content[3] (e.g., news updates from friends) from a user's various SNS accounts (e.g., Facebook, Twitter, Sina Weibo, RenRen), as shown in Fig. 11. To test delay-tolerant downloading, we developed an app called *Timing* in about $2,000$ lines of Java code, in which a file can be decomposed into packets (each being 100 KB) and be downloaded via AppATP. In addition, we deploy *traffic generator* apps that inject packets to the queues in the cloud, with $\lambda \in [0.4, 4.0]$ packets/minute. These test apps enable us to evaluate AppATP in a wide range of application scenarios.

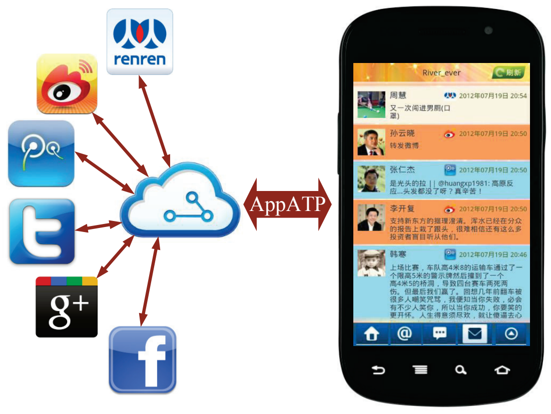


Fig. 11: Apply AppATP in *MixSNS*, a social networking service.

3. AppATP can be employed as a middle-layer OS support that is transparent to upper-layer apps, without requiring modification of mobile apps.

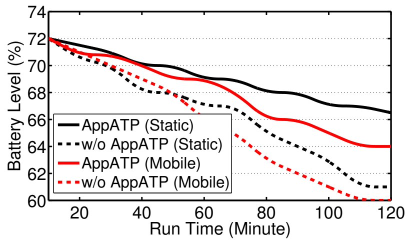


Fig. 12: Total battery consumption for SNS prefetching.

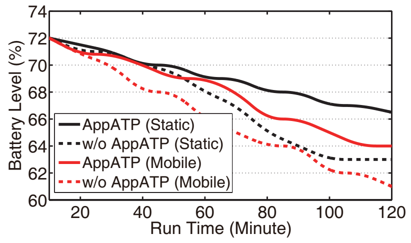


Fig. 13: Battery consumption of data transmissions for SNS prefetching.

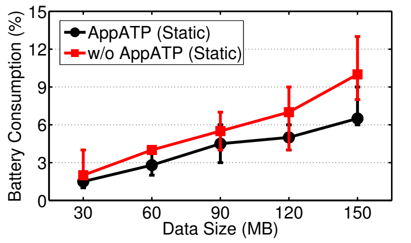


Fig. 14: Battery consumption for file downloading under static scenario.

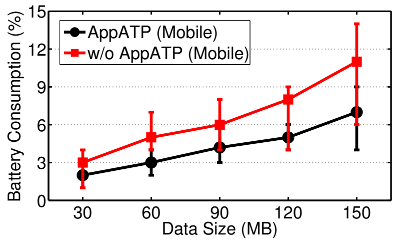


Fig. 15: Battery consumption for file downloading under mobility scenario.

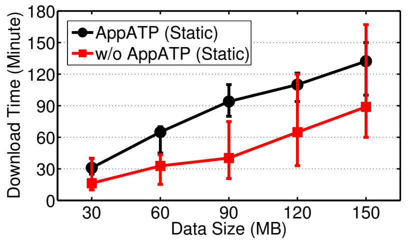


Fig. 16: Completion time for file downloading under static scenario.

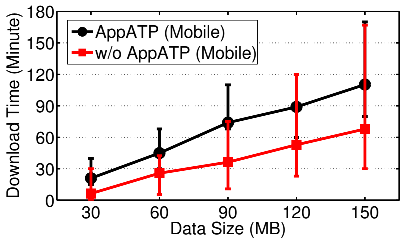


Fig. 17: Completion time for file downloading under mobility scenario.

Real-world experiments are performed on *Samsung Note 2* smartphones running Android, and cover two general user scenarios: the *static* or *mobile* scenario. In the static scenario, we connect the smartphone to a congested WiFi in our lab. Since the WiFi AP is shared by $20-30$ students, the available bandwidth fluctuates frequently. In the mobility scenario, we walk around in a university campus, carrying the smartphones. As the campus is partially covered by public WiFi, there are frequent switches between 3G and WiFi interfaces, leading to changing network conditions as the user moves. To evaluate the energy usage, we close all irrelevant apps and services in the smartphone and record the changes in *battery level* as transmissions happen with or without AppATP.

Note that the cloud-side resource demand of AppATP is small. The complexity of the transmission algorithm is only $O(N)$, and each app requires an average storage of 10 MB only. With an *M1 Small Instance* (with 160 GB instance storage) in Amazon EC2, AppATP can easily support more than $1,500$ users, each running 10 apps, and support $450,000$ users with a *High Storage Instance* (24 2-TB hard disk drives) [25]. The monetary cost for each user is only \$ 0.09 per year [26].

Experimental Results: First, to show how Algorithm 1 works, we run a single traffic generator app in the mobility scenario, with a mean arrival rate $\lambda = 2.0$. In Fig. 9, we plot the queue backlog and downlink bandwidth over time. The transmissions will take place (indicated by the dives in the queue backlog curve) only if the bandwidth is relatively high (e.g., $t = 15, 114, 135$) or if the queue is congested (e .g., $t = 27, 55, 71, 90$). This confirms our remarks on Algorithm 1 in Fig. 3. To evaluate the delay incurred by AppATP, we repeat the test run above for 2 hours, and record the arrival and transmission time of each packet to calculate the average packet delay. As shown in Fig. 10, there is a linear relationship between V and the delay, which confirms the simulation result in Fig. 8 as well as Proposition 1. By tuning V, AppATP can suit the needs of different applications.

We use *MixSNS* mentioned above to evaluate AppATP in prefetch-friendly apps. Since the data arrival rate in SNS apps mainly depends on the number of friends a user has, we create a user account with 200 friends, and let the user run *MixSNS* for 2 hours on two identical smartphones at the same time starting at the same battery level, one phone running AppATP, the other without AppATP. To reflect realistic scenarios, every 20 minutes the app reads the available content for 5 seconds with the screen turned on (note that a user usually keeps the display on while waiting for the data to be retrieved).

In the static scenario, as shown in Fig. 12, the battery level of the phone without AppATP decreases at a much higher rate than with AppATP, consuming 11% battery in 2 hours compared to 6% when using AppATP. In the mobility scenario, AppATP also significantly outperforms the stand alone SNS app as shown in Fig. 12, with 30% energy saving (reducing battery consumption from 12% to 8% in 2 hours). The battery consumption is faster under mobility since the transmission power of cellular interface is higher than WiFi.

Note that the battery saving above not only comes from efficient transmissions, but also because content prefetching eliminates the unnecessary screen-on time while the user actively waits for data retrieval to complete. To evaluate the transmission energy alone without considering display energy, we re-do the above experiments with screen turned off. As shown in Fig. 13, even if the screen energy cost for waiting is removed, the energy consumption for transmitting data via AppATP is still much less than not using AppATP: 45.4% (33.33%) saving for the static (mobility) scenario respectively.

For delay-tolerant application scenarios, we use the downloader app *Timing* to download different sizes of file via AppATP. For each file size, we repeat the experiment for 10 times. As shown in Fig. 14, in static scenario, when the same amount of content is transferred, *Timing* consumes 25%-46% less energy by using AppATP than the android default downloader that continuously downloads the file right after the request. Note that the energy saving becomes significant in the mobility scenario as shown in Fig. 15. Since the bandwidth fluctuates in the presence of 3G and WiFi switchings, AppATP seizes more good transmission opportunities to save energy.

Compared to immediate downloading, AppATP trades delays for energy conservation. To evaluate delay performance, we plot the download completion time of the above experiments in Fig. 16 and Fig. 17. The completion time of AppATP has a higher mean but has a smaller variance, while the completion time variance of immediate downloading is larger in both static and mobility scenarios, especially for large files. If one downloads data in bad connectivity with the default downloader (e.g., the maximum completion time when downloading 30 MB, 120 MB and 150 MB), additional energy is consumed and the completion time is longer than deferring transmissions with AppATP. In these cases, AppATP saves energy consumption without hurting the delay performance.

7 Related Work

To exploit the potential of scheduling transmissions in prefetch-friendly or delay-tolerant apps, IMP [14] presents a cost-benefit analysis to decide when to prefetch data in mobile devices. IMP prefers to transmit when the benefit of prefetching meets the budgets for battery and cellular data usage. In contrast, AppATP focuses on choosing the timing of good wireless connectivity to minimize transmission energy consumption. SALSA [12] proposes to choose an energy-efficient link among available ones using Lyapunov optimization. Instead of solely focusing on link selection, we acknowledge the time-varying nature of wireless connection, and AppATP focuses on choosing the right timing to transmit mobile data. Moreover, while SALSA [12] separately optimizes transmission energy in each delay-tolerant app running on stand-alone mobile devices, AppATP leverages cloud resources to manage data transmissions for multiple prefetch-friendly or delay-tolerant apps collectively, causing the minimum overhead on resource-constrained mobile devices. BreadCrumbs [27], Bartendr [28] and SmartTransfer [29] prefetch or delay mobile data according to wireless network connectivity, but their scheduling strategies rely on the prediction of future network condition to make scheduling decisions. However, unlike wired networks, wireless bandwidth is highly dynamic and changes drastically in short time scales, thus accurate prediction in fast-changing wireless network conditions is impractical. Rather than requiring future prediction, we propose an online algorithm based on detailed energy profiling.

There are two classes of works on how to efficiently combine mobile devices and cloud computing. The first class focuses on offloading resource-intensive tasks from mobile devices to the cloud for energy saving and process speedup. Rather than offloading the app on a fine-grained method/thread level [30], [31], AppATP offloads the transmission management of an app on a component level. The second class of works use the cloud to extend the computing capabilities of mobile devices (Apple's Siri, Google Voice Search, EaaS [32]), or storage (Dropbox, iCloud) and networking (Reflex [33]). In AppATP, the cloud is an extended "cache" for mobile devices, where data are downloaded only during good connectivity.

To improve the energy-efficiency of data transmission for mobile devices, some works consider how to select wireless link from multiple available ones. For example, Virgil [11] estimates AP quality by running probe-based tests and selects the AP with the best connection quality. Seeker [13] uses information available in mesh backhaul for link estimation and selection so as to reduce estimation overhead as compared to probe-based approaches. Context-for-Wireless [34] utilizes history information to estimate network conditions. AppATP can benefit from these complementary studies, as the better link a mobile device has selected, the more opportunities of good connectivity AppATP can seize to transmit data.

8 Conclusions

We present AppATP, an application-layer transmission protocol to achieve energy-efficient data transmissions in mobile devices. By offloading the overhead of management to the cloud, little burden is placed on the mobile device when saving energy via transmission control. AppATP uses Lyapunov optimization to adaptively make transmission decisions according to current network conditions and queue backlogs. We carry out extensive trace-driven simulations and a real-world implementation on Amazon EC2, and show that AppATP can have huge energy saving on mobile devices.

Acknowledgment

The research was supported in part by a grant from National Basic Research Program (973 program) under grant No.2014CB347800.

References

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017. [Online]. Available: http://www.cisco.com/

[2] Flurry Five-Year Report, http://blog.flurry.com/?Tag=App+Usage.

[3] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer, 43(4), 51–56, 2010.*

[4] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing," in *Proc. of IEEE Infocom*, 2013.

[5] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li, "eTime: energy-efficient transmission between cloud and mobile devices," in *Proc. of IEEE Infocom Mini-conference*, 2013.

[6] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proc. of IMC*, 2009.

[7] J. Huang, F. Qian, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. of ACM MobiSys*, 2012.

[8] Use Wi-Fi Instead of 3G to Save Android Battery Life. [Online]. Available: http://goo.gl/HLND7

[9] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, "Gearing Resource-Poor Mobile Devices with Powerful Clouds: Architectures, Challenges and Applications," *IEEE Wireless Comm. Mag., 20(3), 2013.*

[10] M. Neely, "Stochastic network optimization with application to communication and queueing systems," *Morgan & Claypool Publishers*, 2010.

[11] A. Nicholson, Y. Chawathe, M. Chen, B. Noble, and D. Wetherall, "Improved access point selection," in *Proc. of ACM MobiSys*, 2006.

[12] M. Ra, J. Paek, A. Sharma, R. Govindan, M. Krieger, and M. Neely, "Energy-delay tradeoffs in smartphone applications," in *ACM MobiSys*, 2010.

[13] D. Gupta, P. Mohapatra, and C. Chuah, "Seeker: A bandwidth-based association control framework for wireless mesh networks," *Wireless Networks*, 2011.

[14] B. Higgins, J. Flinn, T. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed Mobile Prefetching," in *Proc. of ACM MobiSys*, 2012.

[15] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. of eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2010.

[16] A. Pathak, Y. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proc. of ACM EuroSys*, 2012.

[17] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. Neely, "Data centers power reduction: a two time scale approach for delay tolerant workloads," in *Proc. of IEEE Infocom*, Mar. 2012.

[18] Z. Zhou, F. Liu, , Y. Xu, R. Zou, H. Xu, J. Lui, and H. Jin, "Carbon-aware Load Balancing for Geo-distributed Cloud Services," in *Proc. of IEEE MASCOTS*, August 2013.

[19] Z. Zhou, F. Liu, H. Jin, B. Li, and H. Jiang, "On Arbitrating the Power-Performance Tradeoff in SaaS Clouds," in *Proc. of IEEE INFOCOM*, 2013.

[20] W. Deng, F. Liu, H. Jin, C. Wu, and X. Liu, "Multigreen: Cost-minimizing multi-source datacenter power supply with online control," in *Proceedings of the Fourth International Conference on Future Energy Systems*, ser. e-Energy '13, 2013.

[21] W. Deng, F. Liu, H. Jin, and C. Wu, "Smartdpss: Cost-minimizing multi-source power supply for datacenters with arbitrary demand," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, 2013.

[22] M. Neely, E. Modiano, and C. Rohrs, "Dynamic power allocation and routing for time-varying wireless networks," *IEEE JSAC*, 2005.

[23] D. Hardt, "The OAuth 2.0 authorization framework," 2012.

[24] J. Guo, F. Liu, D. Zeng, J. C. Lui, and H. Jin, "A cooperative game based allocation for sharing data center networks," in *IEEE INFOCOM*, 2013.

[25] F. Xu, F. Liu, H. Jin, and A. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions," *Proceedings of the IEEE*, vol. 102, no. 1, pp. 11–31, Jan 2014.

[26] Amazon EC2 Instance Types. [Online]. Available: http://aws.amazon.com/cn/ec2/instance-types/

[27] A. Nicholson and B. Noble, "Breadcrumbs: forecasting mobile connectivity," in *Proc. of ACM MobiCom*, 2008.

[28] A. Schulman, N. Vishnu, R. Ramachandran, S. Neil, D. Pralhad, G. Calvin, J. Kamal, and N. Venkata, "Bartendr: a practical approach to energy-aware cellular data scheduling," in *Proc. of MobiCom*, 2010.

[29] Y. Wang, X. Liu, and A. Nicoara, "SmartTransfer: Transferring Your Mobile Multimedia Contents at the "Right" Time," in *Proc. of NOSSDAV*, 2012.

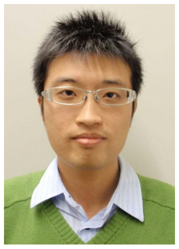
[30] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. of ACM MobiSys*, 2010.

[31] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. of EuroSys*, 2011.

[32] M. Altamimi, R. Palit, K. Naik, and A. Nayak, "Energy-as-a-Service (EaaS): On the efficacy of multimedia cloud computing to save smartphone energy," in *Proc. of IEEE Cloud*, 2012.

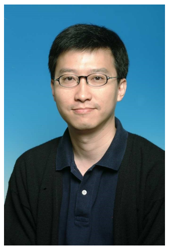
[33] Z. Liu, Y. Feng, and B. Li, "Socialize spontaneously with mobile applications," in *Proc. of IEEE Infocom*, 2012.

[34] A. Rahmati and L. Zhong, "Context-based network estimation for energy-efficient ubiquitous wireless connectivity," *IEEE Transactions on Mobile Computing*, 2011.



Fangming Liu is an associate professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China; and he is awarded as the CHUTIAN Scholar of Hubei Province, China. He is the Youth Scientist of National 973 Basic Research Program Project on Software-defined Networking (SDN)-based Cloud Datacenter Networks, which is one of the largest SDN projects in China. Since 2012, he has also been invited as a StarTrack Visiting Young Faculty in Microsoft Research Asia (MSRA), Beijing. He received his B.Engr. degree in 2005 from Department of Computer Science and Technology, Tsinghua University, Beijing; and his Ph.D. degree in Computer Science and Engineering from the Hong Kong University of Science and Technology in 2011. From 2009 to 2010, he was a visiting scholar at the Department of Electrical and Computer Engineering, University of Toronto, Canada. He was the recipient of two Best Paper Awards from IEEE GLOBECOM 2011 and IEEE CloudCom 2012, respectively. His research interests include cloud computing and datacenter networking, mobile cloud, green computing and communications, software-defined networking and virtualization technology, large-scale Internet content distribution and video streaming systems. He is a member of IEEE and ACM, as well as a member of the China Computer Federation (CCF) Internet Technical Committee. He has been a Guest Editor for IEEE Network Magazine, an Associate Editor for Frontiers of Computer Science, and served as TPC for IEEE INFOCOM 2013-2015, ICNP 2014, ICDCS 2015, and ACM Multimedia 2014.

Peng Shu received his BS and MS degree in School of Computer Science and Technology, Huazhong University of Science and Technology, China. He is currently working at Synopsys, Wuhan, China. His research interests focus on cloud computing and wireless mobile applications.



John C. S. Lui is currently a professor in the Department of Computer Science and Engineering at The Chinese University of Hong Kong. He received his Ph.D. in Computer Science from UCLA. His current research interests are in communication networks, network/system security (e.g., cloud security, mobile security, etc.), network economics, network sciences (e.g., online social networks, information spreading, etc.), cloud computing, large scale distributed systems and performance evaluation theory. John serves in the editorial board of IEEE/ACM Transactions on Networking, IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, Journal of Performance Evaluation and International Journal of Network Security. John was the chairman of the CSE Department from 2005 to 2011. He received various departmental teaching awards and the CUHK Vice-Chancellors Exemplary Teaching Award. He is also a corecipient of the IFIP WG 7.3 Performance 2005 and IEEE/IFIP NOMS 2006 Best Student Paper Awards. He is an elected member of the IFIP WG 7.3, Fellow of ACM, Fellow of IEEE and Croucher Senior Research Fellow.