This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing

Quality-Aware Traffic Offloading in Wireless Networks

Wenjie Hu, *Student Member, IEEE,* and Guohong Cao, *Fellow, IEEE*

Abstract—In cellular networks, due to practical deployment issues, some areas have good wireless coverage while others may not. This results in significant throughput (service quality) difference between wireless carriers at some locations. We first analyze the factors that affect the service quality and then validate the existence of service quality difference between different carriers via extensive measurements. To deal with this problem, a mobile device (node) with low service quality can offload its data traffic to nearby nodes with better service quality through Device-to-Device interfaces, such as WiFi direct, to save energy and reduce delay. To achieve this goal, we propose a *Quality-Aware Traffic Offloading (QATO)* framework to offload network tasks to neighboring nodes with better service quality. QATO can identify neighbors with better service quality and motivate nodes to help each other using incentive schemes. To validate our design, we have implemented QATO on Android platform and have developed a web browser and a photo uploader on top of it. Experimental results show that QATO can significantly reduce energy and delay for both data downloading and uploading. Through trace-driven simulations, we also show that all users can benefit from data offloading in the long run.

Index Terms—Data offloading; Energy Saving; Cellular Networks; Smartphone

1 Introduction

In cellular networks such as 3G, 4G and LTE, mobile devices are served by Base Stations (BSs) which cover a large area (about 1-2 miles). Due to practical deployment issues, some areas have good coverage while others may not. As a result, the wireless signal strength of a mobile device varies based on its location. Moreover, the data throughput in an area also depends on the number of people in that area and the backhaul network of the wireless carrier [26]. When the service quality (in terms of throughput) is low, it takes longer time to transmit the same amount of data and consumes more energy.

Some existing work has addressed the service quality difference at different locations. Schulman *et al.* proposed to defer data transmission to save energy when the service quality is low [29]. However, this solution only works when it is known that the user will quickly move to a location with better service quality. There are also solutions on offloading cellular traffic to WiFi network to save energy and improve service quality [22]. However, WiFi access may not always be available.

In this paper, we address the service quality difference from a different perspective. Through theoretical analysis, we show that the service quality varies in different locations due to the received signal strength from the BS. Through extensive measurements, we observe that mobile devices (nodes) within an area may have different service quality and thus different throughput (e.g., a node may consume much more energy and delay to download the same amount of data), especially when different wireless carriers are used. To deal with this problem, a node with low service quality can offload its traffic to the node with better service quality through Device-to-Device (D2D) interfaces such as WiFi direct, to save energy and reduce delay. Based on this finding, we propose a *Quality-Aware Traffic Offloading (QATO)* framework, where nodes with low service quality may offload their data traffic to those with better quality via the WiFi direct interface. QATO can identify neighbors with better service quality through service discovery, and offload traffic to them. We also provide incentive mechanisms to motivate nodes to help each other.

We have implemented the QATO framework on Android platforms. To evaluate its performance, we developed two applications based on QATO: a Web browser which is used to evaluate the performance of download offloading and a photo uploader which focuses on upload offloading. Experimental results show that QATO can reduce energy by 38% in downloading and 70% in uploading, and reduce delay by 45% in downloading and 88% in uploading. Trace-driven simulations are used to evaluate the performance at a larger scale, and the results show that all nodes can save energy and reduce delay in the long run. Our contributions are as follows.

- We conduct extensive measurements to show the existence of service quality difference between different carriers and then introduce the idea of leveraging the service quality difference among nearby nodes to save energy and reduce delay.
- We design a quality-aware traffic offloading framework to automatically detect neighbors willing to help others and offload traffic to such nodes with better service quality, and consider many practical issues. Also, we design proper incentive mechanisms to encourage users to help each other.
- We implement the QATO framework on the Android platform and develop two applications on top of it. We also use the real testbed to verify the effectiveness of QATO framework.

The rest of the paper is organized as follows. Section 2 introduces the background of different cellular networks and their energy and delay model. Section 3 provides the motivation for quality-aware traffic offloading. We present the design and im-

- *Wenjie Hu and Guohong Cao are with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802. E-mail: {wwh5068,gcao}@cse.psu.edu.*

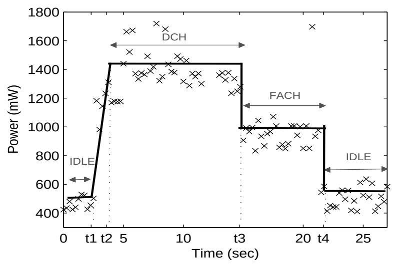


Fig. 1. The power level of using the UMTS cellular interface (with screen on)

TABLE 1
Mobile Devices and Network Types

Device	Provider	Net[1]	Net[2]
Samsung Galaxy S3	Carrier 1	HSPA+	LTE
Samsung Galaxy S4	Carrier 2	LTE	LTE

[1] Network in City 1; [2] Network in City 2

plementation of the traffic offloading framework in Section 4 and Section 5, respectively. The performance of QATO is evaluated in Section 6. Section 7 discusses related work, and Section 8 concludes the paper.

2 Preliminaries

In this section, we first give a short description of different cellular networks, and then introduce the energy and delay model.

2.1 UMTS, HAPA+ and LTE Network

UMTS (the Universal Mobile Telecommunication System) is a popular 3G standard developed by 3GPP. It provides a maximum bit rate of 384 Kbps to a single user at its first version, release 99. To support higher data rate, High Speed Downlink Packet Access (HSDPA) and High Speed Uplink Packet Access (HSUPA) are added to UMTS to improve the downlink and uplink speed. HSDPA and HSUPA are then merged and the enhanced version is called HSPA+, also referred to as 4G. In HSPA+, enhanced techniques, such as 64 QAM and Multiple-Input Multiple-Output (MIMO), are used to increase the data rate up to 84 Mbps [33]. LTE (the Long-Term Evolution) is the latest extension of UMTS, and it enhances both the radio access network and the core network. The core network architecture of LTE is based on all-IP network and can support other non-3GPP radio access networks such as WiMAX and CDMA2000, which enables these networks to adopt LTE as their future radio access network. LTE can provide much higher bandwidth than 4G [15].

2.2 Energy and Delay Model

The power model of a typical data transmission in UMTS 3G networks is shown in Fig. 1. Initially, a node stays at the IDLE state, which consumes little power. When a data request arrives, it promotes to the data transmission state (DCH), where data can be sent/received at high speed. After a data transmission, it stays at the TAIL state (FACH) for a period of time, in case another data request arrives soon.

Since UMTS 3G/4G/LTE have close relationships, their power model during data transmission is also similar. Thus we use a generalized power model for all these networks. The power of cellular interface has three states: *promotion*, *data transmission* and *tail*, and the power consumption of these states are denoted as P_{pro}, P_{cell} and P_{tail}, respectively. The energy consumption of a task in cellular network can be modeled as follows. Suppose task T_i arrives at t_i with data size d_i, and the most recent task on the same node is T_j. Then the time interval between the tail end of T_j and the start time of T_i is denoted as Δt, and $\Delta t = t_i - t_j - d_j/r_{cell} - t_{pro}$ if T_j starts to transmit when the cellular interface is on idle state, and $\Delta t = t_i - t_j - d_j/r_{cell}$ if T_j starts to transmit on high power state (data transmission or tail state), where r_{cell} is the data throughput of the given node. There are three cases to compute T_i's power consumption depending on Δt. 1) If Δt is larger than the tail timer t_{tail}, i.e., the cellular interface is in IDLE state before T_i arrives, then T_i will consume extra promotion and tail energy, besides the data transmission energy. 2) If Δt is smaller than t_{tail} but bigger than 0, there is part of tail energy but no promotion energy. 3) If Δt is smaller than 0, T_i will be overlapped with T_j for some time, and there will be no additional tail energy.

$$E^i_{cell}(T_j) = \begin{cases} P_{pro} \times t_{pro} + P_{cell} \times \frac{d_i}{r_{cell}} + P_{tail} \times t_{tail}, & \text{if } \Delta t > t_{tail} \\ P_{cell} \times \frac{d_i}{r_{cell}} + P_{tail} \times \Delta t, & \text{if } 0 < \Delta t \leq t_{tail} \\ P_{cell} \times \max\{\Delta t + \frac{d_i}{r_{cell}}, 0\}, & \text{Otherwise} \end{cases} \quad (1)$$

The delay to complete a task is also related with Δt. If Δt is smaller than t_{tail}, the delay is just the data transmission time. Otherwise, the delay should include additional promotion delay, as shown in Eq. 2.

$$D^i_{cell}(T_j) = \begin{cases} d_i/r_{cell} + t_{pro}, & \text{if } \Delta t > t_{tail} \\ d_i/r_{cell}, & \text{Otherwise} \end{cases} \quad (2)$$

3 The Motivation of Quality-aware Traffic Offloading

In this section, we first identify the reasons behind the difference in service quality (in terms of throughput) between different carriers, and then conduct extensive measurements to verify this difference and quantify its effects on energy and delay. Finally, we show the benefits of quality-aware traffic offloading.

3.1 Throughput Difference Between Carriers

In cellular networks, BSs are used to provide services for mobile devices within their coverage. The BS may be several miles away from the mobile device and the wireless signal between them may be blocked by multiple objects and may be affected by multipath interference. A commonly used metric to measure the signal quality is the signal to noise ratio (SNR), which affects the throughput in two ways. First, SNR determines the channel capacity. According to the Shannon-Hartley theorem [3], the maximum rate a channel can transmit is determined by Eq. 3.

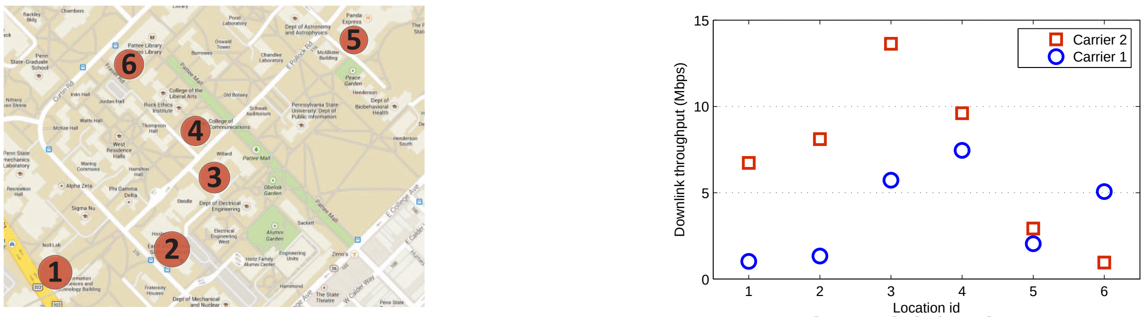
$$r_{max} = B \log_2(1 + SNR) \quad (3)$$

where r_{max} is the channel rate, and B is the frequency bandwidth of the channel, which is generally fixed given a specific wireless carrier. Second, SNR affects the packet loss rate. When SNR is below some threshold (e.g., 20 dB), the packet loss rate will increase sharply and few packets can be received successfully [34].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing

TABLE 2
Power consumption of different networks

	State	Power (mW)	Duration (s)	Download throughput (Mbps)	Upload throughput (Mbps)
HSPA+ (Carrier 1)	Promotion	1422.2 ± 34.1	2.3 ± 0.5	-	-
	Data	1990.9 ± 44.2	-	4.5 ± 1.3	1.3 ± 0.2
	Tail	1622.6 ± 39.6	11.4 ± 1.4	-	-
LTE (Carrier 1)	Promotion	1214.8 ± 24.3	0.25 ± 0.4	-	-
	Data	1865.8 ± 25.6	-	51.11 ± 16.9	17.9 ± 5.1
	Tail	1125 ± 22.3	11.5 ± 0.56	-	-
LTE (Carrier 2)	Promotion	1567.9 ± 47.8	0.34 ± 0.04	-	-
	Data	2224.7 ± 53.1	-	15.4 ± 5.5	5.3 ± 2.5
	Tail	1757.5 ± 97.5	3.37 ± 0.08	-	-
WiFi direct	Data	1323.6 ± 13.9	-	29.5 ± 1.2	29.5 ± 1.2



(a) Locations (b) Downlink throughput

Fig. 2. Downlink throughput in different locations. There are coverage blind spots in location 1 and location 2 for Carrier 1, and location 6 for Carrier 2, which motivates the use of traffic offloading to improve service quality.

Besides *SNR*, the number of users served by the BS can also affect the throughput. Suppose there are M active users served by the same BS on the same channel, the maximum channel rate for one user would be no more than r_{max}/M. Thus, in a crowded area such as New York city, the data rate may be much lower even when the wireless signal is strong.

Putting them together, the throughput of a device is affected by *SNR* and the number of active users in the BS. For different wireless carriers at the same location, these factors are different and thus their throughput varies.

3.2 Measuring the Quality Difference between Carriers

We use two types of smartphones (Samsung Galaxy S3 and S4) to measure the throughput of two wireless carriers (denoted as *Carrier 1* and *Carrier 2*) in two cities (denoted as *City 1* and *City 2*). A brief description of the devices and networks is shown in Table 1.

To measure the throughput, we ported *iperf* to smartphones, added timestamp to record the start and end of a data transmission, and extended it to support WiFi direct. With iperf, the smartphone establishes a TCP connection to our backend server and measures the downlink and uplink throughput for 30 seconds. The throughput measurement has been done for two months in different locations (inside and outside of buildings in different cities), at different time. A total number of more than 300 measurement results are collected. Besides the throughput, the power consumption during data transmission is also measured. We use the Monsoon power monitor to provide power supply for smartphones, which provides constant voltage and measures the current at a rate of 5000Hz. Based on the power measurement trace and the start/end time from the iperf trace, we can get the power consumption at different network state. The results are shown in Table 2, where the power value is measured as the whole phone's power when the screen is on. Our measurement is different from previous work (e.g., [15]) from three perspective. First, our measurement is based on more recent LTE network deployment, which reflects the significant technology advancement in recent several years. Second, our measurement considers multiple smartphones, which shows the impact of phone models on network throughput and power consumption. Third, we consider the difference in throughput and power consumption between different carriers at the same locations, which is not considered in other work.

3.2.1 *Micro Perspective: Coverage Blind Spots of Different Carriers*

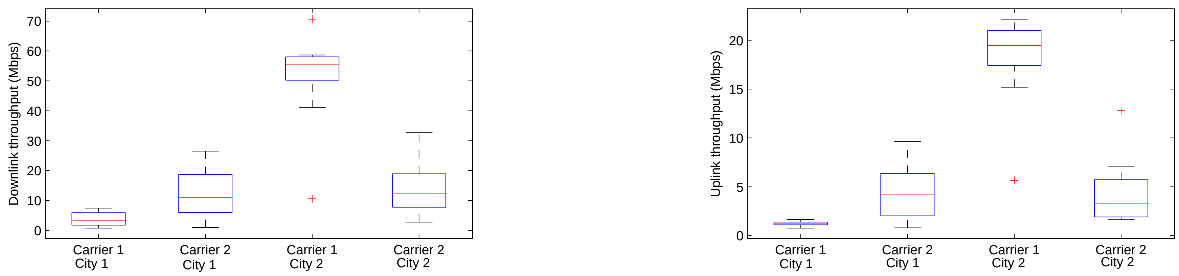
Within the coverage of a BS, the data rate within an area varies greatly due to many reasons, such as the distance from the BS, obstacles on the way, interference from other devices, etc. It is common that one carrier has some coverage blind spots at some locations, which means that the throughput is extremely low and the quality of experience is poor. This problem is especially worse in indoor environments.

We picked 6 popular locations in our university, including Lab, library, classroom, and Cafeteria, as shown in Fig. 2(a). In each location, we measured the data throughput of different carriers at the same time. The comparison results are shown in Fig. 2(b). As can be seen, Carrier 1 has extremely low throughput in locations 1 and 2 but carrier 2 has much better service quality. The situation is just the opposite in location 6. In these blind spots, there is strong motivation for nodes to offload traffic to neighbors with better service quality.

3.2.2 *Macro Perspective: Quality Complementary between Carriers*

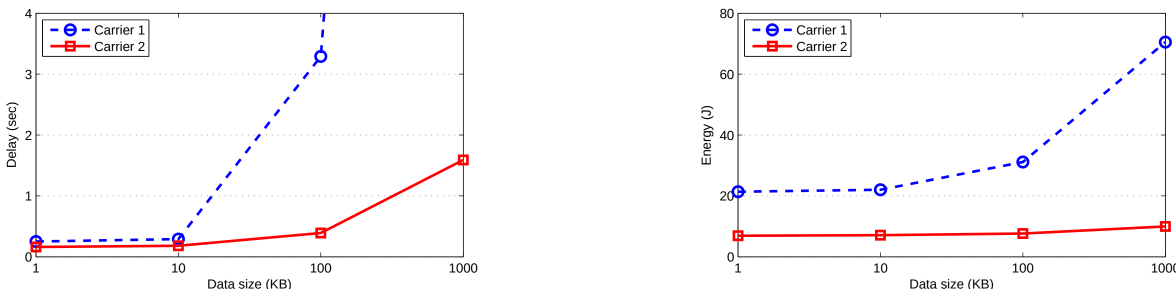
Different carriers have different priorities when deploying cellular networks in different cities. One may provide better service quality

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing



(a) Downlink (b) Uplink

Fig. 3. Throughput of different carriers in City 1 and City 2. Carrier 2 provides better service quality in City 1 while Carrier 1 provides better service quality in City 2.



(a) Delay comparison (b) Energy comparison

Fig. 4. Energy and delay comparison between different carriers with different throughput.

in one city but lower quality in another city, while the reverse is true for another carrier. To verify this hypothesis, we collect data throughput of two carriers in two cities, as shown in Table 1. In each city, we collect the throughput in multiple locations at different time. The downlink and uplink throughput of both carriers in these two cities are shown in Fig. 3(a) and Fig. 3(b), respectively. In the boxplot figure, the middle line of a box indicates the median, and the lower and upper side of the box are the first (25%) and third (75%) quartile, which are denoted by $Q1$ and $Q3$. The outliers are values outside $1.5\times(Q3-Q1)$ range above $Q3$ or below $Q1$.

Both downlink and uplink throughput show the same trend. Carrier 1's HSPA+ network provides lower service quality than Carrier 2's LTE network while Carrier 1's LTE network outperforms that of Carrier 2. Thus we have two findings. First, there is high throughput difference between carriers, which can be as much as *eight times*. Second, the service quality provided by different carriers varies in different cities and complements each other, which provides opportunity and motivation to help each other. For example, suppose a group of commuters using these two wireless carriers periodically travel between these two cities, then users of Carrier 2 may share their services to users of Carrier 1 in City 1, and utilize the service from Carrier 1 in City 2. Thus, with data offloading, all users can benefit in the long run.

3.2.3 Delay and Energy Comparisons

Under different service quality, the data access delay and the power consumption to accomplish a network task are different. To measure this difference, we use the settings in City 1 as mentioned in Table 1 to download a given size of file from our server. The delay is the time from the start of downloading a file to the time when the file is completely received. The energy is measured as the whole phone's power consumption during downloading (including tail energy) when the screen is on.

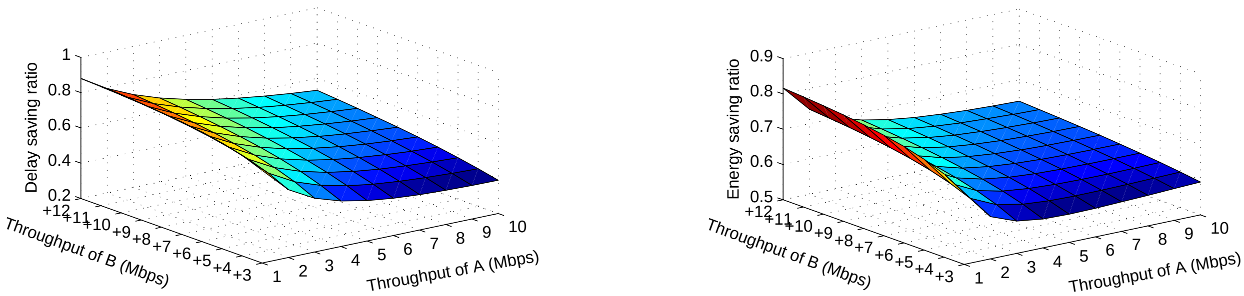
Figure 4 shows the results. As can be seen in Fig. 4(a), the data access delay is similar for both carriers when the data size is small. As the data size increases, the access delay of Carrier 1 increases much faster than that of Carrier 2. This result is consistent with our experience in real life; i.e., when updating emails or browsing simple webpages, LTE feels similar to 3G network; however, when watching movies, LTE significantly outperforms 3G. This result indicates that offloading large data to nodes with better quality can significantly reduce the delay.

Figure 4(b) compares the energy of data transmission with different service quality. Since Carrier 1 takes longer time to transmit the data, and has larger promotion and tail energy, it consumes much more energy than Carrier 2. Fortunately, the energy consumption does not increase linearly with the data size. For example, when the data size increases from 10KB to 1000KB, the energy only increases 3.2 times for Carrier 1 and 1.4 times for Carrier 2. It means that aggregating a large amount of data on a node with higher throughput does not increase the energy too much.

3.3 The Benefit of Traffic Offloading

In order to see the benefit of traffic offloading, we show some numerical results based on an example where user A (with low throughput) offloads 2MB of data to user B (with high throughput). Suppose E_{local} is the energy consumed by user A to transmit data using its own cellular interface, and E_{remote} is the energy consumed by both users when A offloads the traffic to B through WiFi direct. Then, the energy saving ratio can be computed as $(E_{local}-E_{remote})/E_{local}$, and the delay saving ratio can be computed as $(D_{local}-D_{remote})/D_{local}$. We adjust the throughput

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing



(a) Delay saving ratio (b) Energy saving ratio

Fig. 5. Energy and delay saving with traffic offloading. The throughput of *B* indicates the throughput increase compared to the corresponding value of *A*.

of both users and show the saved energy and delay in Fig. 5. Here the power parameters of user *A* and user *B* are based on those of Carrier 1 (HSPA+) and Carrier 2 (LTE) as listed in Table 2, respectively. As shown in the figure, when there is large throughput difference between the users, traffic offloading can save more energy and delay. Also, when one of them has low throughput, it is more beneficial to use traffic offloading.

4 QATO Design

In this section we introduce the design of our data offloading framework QATO. We first give an overview of QATO and then describe its major components.

4.1 QATO Overview

Consider that a group of users stay together for a relatively long time, such as in a Lab or on a commuting bus from one city to another. They use different wireless carriers and are willing to share data services with others to trade for better service from others at a later time. In such scenarios, QATO enables phones to offload their network tasks to a neighboring node with better service quality. Here a network task means an independent task to fulfill one user request, such as downloading one webpage or uploading one photo, which may contain lots of data packets.

The architecture of QATO is shown in Fig. 6. In the original smartphone system, all network tasks are buffered in the local queue and then scheduled based on the FIFO (First In First Out) order. Here a network task means an independent task to fulfill one user request, such as downloading one webpage or uploading one photo, which may contain lots of data packets. With QATO, the network tasks are guided into the offload engine module first. By taking into account the local network information and neighbor network information collected by the service discovery module, the offload engine module determines whether to offload the network tasks and to which node to offload. The data transmission module maintains a local task queue and a remote task queue, and properly schedule them to reduce energy and delay. Since users may be selfish, we also design a credit based mechanism to motivate users to share their cellular service. The credit manager module manages credits; i.e., collecting credits from nodes using the service and pay credits to nodes providing the service.

4.2 Service Discovery

There has been some existing research on detecting nearby users recently; however, most of them are based on Bluetooth [14].

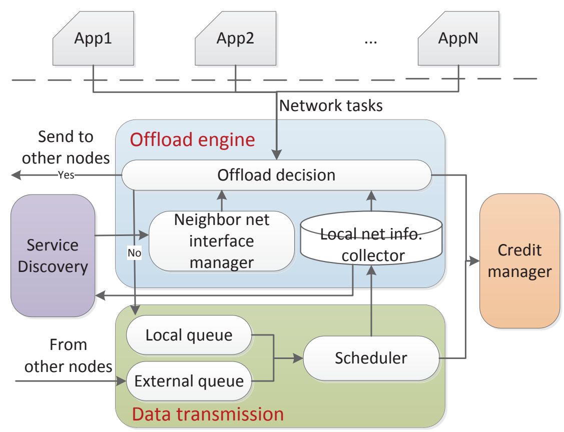


Fig. 6. QATO architecture

Since the communication range of Bluetooth is short and its throughput is low, another D2D interface, called WiFi direct, is widely used to detect neighbors in recent years [6] and it is recommended in the 3GPP proximity service (ProSe) standard [4, 24]. However, there are few real implementations. Moreover, in our case, we need to know which neighbors support QATO, i.e., are willing to offload traffic for others. Thus, simply detecting neighbors using WiFi direct is not enough. To solve this problem, we leverage the DNS (Name Domain System) based service discovery (RFC 6763) to find neighbors. It allows nodes to discover neighbors supporting a specific service using the WiFi direct interface directly, without the support of central servers and access points.

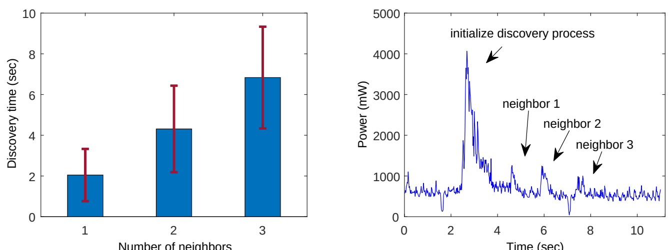
Android begins to support DNS-based service discovery since Android 4.1 (Jelly Bean), and it supports DNS-SD to be deployed on WiFi direct interface. On each node, we register “QATO” as a service, with “_http._tcp” as the service type, and a local port assigned to this service. After successful registration, the node will be able to respond to the “QATO” requests from neighbors. Note that the user can also turn off the “QATO” service if he does not have traffic to offload. In the response, it also includes its IP address and port number. Based on such information, two nodes can connect with each other via the WiFi direct interface and exchange the network quality information. The network quality information is organized as an XML based profile, which contains wireless network information such as the type of service, the signal strength and the downlink/uplink throughput, and task

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing

TABLE 3
Notations

Notations	Descriptions
r_{D2D}, P_{D2D}	Throughput and power of the D2D network
t_{pro}, P_{pro} *	Promotion delay and power of cellular network
t_{tail}, P_{tail} *	Tail time and power of cellular network
r_{cell}, P_{cell} *	Throughput and power of cellular network
λ_l, λ_r *	Data arriving rate of local and remote task queue
S_r, $\overline{S_r}$ *	Total and average data size in remote task queue
Γ *	Data size threshold for scheduling remote tasks

*: notations with superscript p indicate the corresponding value of the proxy node



(a) Time (b) Power

Fig. 7. The time duration and power consumption of the service discovery process in WiFi direct

related information such as the local task arriving rate, remote task arriving rate and total remote task size. Some frequently used notations are listed in Table 3.

For each node, it collects a list of network quality profiles from neighbors through service discovery. Then, it builds a *proxy list*, i.e., the potential nodes for offloading, which contains neighbors with much higher cellular throughput than itself.

4.2.1 Cost Analysis

The service discovery process consumes extra energy, and we quantify it by measuring the time duration and power consumption. The time duration is related to the number of neighbors, as shown in Fig. 7(a). Generally speaking, discovering one neighbor takes about two seconds. The power consumption of service discovery is much higher than the idle state. Fig. 7(b) shows a power consumption trace of discovering three neighbors. The discovery process lasts for 5.77 seconds and consumes 7.06J of energy, which is similar to uploading 590KB of data with Carrier 1's HSPA+ network.

Frequently executing service discovery consumes too much energy, but many neighbors may not be detected with low discovery frequency [14]. To save energy, the service discovery process in QATO is only started when a node's proxy list is empty, and the discovery period should be adjusted considering the real context, which is discussed later in Section 4.4.3. The service discovery process will also be started when a node in the selected proxy list disappears or when the proxy's data throughput becomes worse than itself.

4.3 Data Transmission Schedule

In QATO, a node maintains two task queues, local task queue Q_l and remote task queue Q_r, to store the network tasks generated locally and received from neighbors, respectively. During data transmission schedule, i.e., task schedule, we consider two factors. First, local tasks should not be affected by remote tasks. Second,

Algorithm 1 Task Schedule on Proxy Node

```
function TASKSCHDULE(Q_l, Q_r)
    S_r ← 0, TailEnd ← 0
    order all tasks by task time
    for each task T_i ∈ Q_l ∪ Q_r do
        if T_i ∈ Q_l then
            EXECUTETASK(T_i)           /* Get T_i by cellular interface */
            Q_l ← Q_l \ T_i     /* Remove T_i from local task queue Q_l */
            TailEnd ← t_i + t_pro + d_i/r_cell + t_tail
            EXECUTEREMOTETASK_CASE1(Q_r, TailEnd)
        else
            S_r ← S_r + d_i
            if S_r > Γ then
                EXECUTEREMOTETASK_CASE2(Q_r, S_r)
                S_r ← 0
            end if
        end if
    end for
end function
function EXECUTEREMOTETASK_CASE1(Q_r, TailEnd)
    for T_j ∈ Q_r do
        if t_j < TailEnd then
            EXECUTETASK(T_j)
            TailEnd ← t_j + d_j/r_cell + t_tail
            Q_r ← Q_r \ T_j
            S_r ← S_r − d_j
        end if
    end for
end function
function EXECUTEREMOTETASK_CASE2(Q_r, S_r)
    for T_j ∈ Q_r do
        if S_r > 0 then
            EXECUTETASK(T_j)
            Q_r ← Q_r \ T_j
            S_r ← S_r − d_j
        end if
    end for
end function
```

remote tasks should be scheduled only when their introduced tail energy is negligible. To solve this problem, we design a scheduling algorithm for the proxy node, as shown in Algorithm 1. A local task is scheduled when it is generated since it has higher priority. For a remote task, it is scheduled based on the following two cases to save energy:

- **Case 1:** A remote task is executed when the cellular interface is already in the data transmission state, either triggered by local tasks or by previous remote tasks. In this case, we record the tail ending time. After scheduling a task, the tail ending time is also extended.
- **Case 2:** Remote tasks are executed in a bunch when the accumulated data size is larger than a threshold Γ. This ensures that remote tasks are scheduled when there is no pending local task.

An example is shown in Figure 8. Remote task R_1 arrives first but is not scheduled immediately. Local task L_1 is scheduled when it is generated. After that, R_1 is scheduled as the cellular interface is on the data transmission state, as illustrated in Case 1. Similarly, R_2 is also scheduled. When R_3 arrives, the cellular interface has moved to the IDLE state, so it waits for future tasks. When R_6 arrives, the accumulated remote tasks are more than Γ, and the four buffered tasks are scheduled together, as discussed in Case 2.

The selection of Γ affects energy and delay. Larger Γ means more tasks are scheduled together to amortize the tail energy, and therefore reducing the energy cost but increasing the delay. On

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing

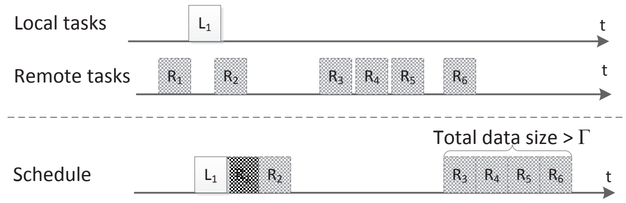


Fig. 8. The scheduling of local tasks and remote tasks

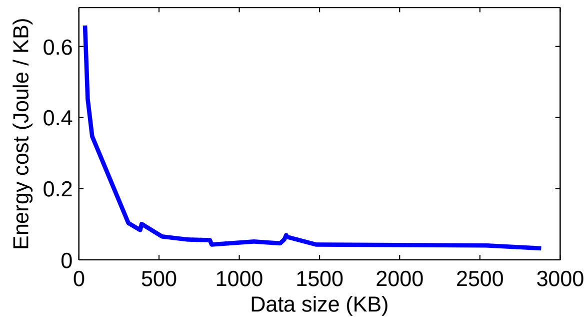


Fig. 9. An example of the selection of Γ

the other hand, smaller Γ can reduce the delay but increase the energy cost. Due to the throughput difference between uplink and downlink, Γ can be different according to the traffic direction.

To set up Γ, we compute the energy cost for transmitting a given amount of data according to the network type and traffic direction, and show the relationship between Γ and energy cost. For example, we measure the energy consumption of uploading different amount of data under Carrier 1 in City 1, and the results are shown in Fig. 9. As can be seen, when the data size is larger than 500KB, there is a big drop in the energy cost. Thus, it is better to set Γ bigger than 500KB in this case.

We also provide users with options to adjust Γ. From the users' perspective, a larger Γ helps to save energy, at the cost of increasing the delay for remote tasks and reducing the chance of serving more remote tasks to earn credit (see Section 4.5). On the other hand, if Γ is small, which can be 0 in the extreme case, the remote tasks can be scheduled immediately. Proxies can select the proper Γ value considering the remaining energy and the willingness to earn credit.

4.4 Offload Engine

As mentioned before, each node maintains a proxy list after service discovery. When a network task arrives, the offload engine will determine whether to offload the task. If so, it selects the proxy node p from the list considering both energy and delay.

4.4.1 Energy Consideration

Suppose task T_i is generated at time t_i, with data size d_i. If this task is executed locally, the energy consumption E^i_{local} can be computed using Eq. 1. Otherwise, if this task is offloaded to p, the data transmission energy should be the data transmission energy on node p, plus the additional energy of the D2D interface. As mentioned in Section 4.3, there are two cases to schedule remote task T_i. In Case 1, there is no additional promotion and tail energy; in Case 2, T_i shares part of these energy proportional to its size. As we are not sure when the future tasks will be scheduled, the total energy is computed based on the worst case, as shown in Eq. 4, where r_{D2D} and P_{D2D} are the throughput and power of the D2D interface, i.e., WiFi direct in this paper.

$$E^{i,p}_{remote} = \frac{d_i}{r_{D2D}} \times P_{D2D} + \frac{d_i}{r^p_{cell}} \times P^p_{cell} + \frac{d_i}{\Gamma^p} \times (P^p_{tail} \times t^p_{tail} + P^p_{pro} \times t^p_{pro}) \quad (4)$$

4.4.2 Delay Consideration

If task T_i is executed locally, the delay D^i_{local} can be computed using Eq. 2. When offloaded to proxy node p, the delay contains four parts: the time to execute the first task in the remote queue, queue delay, the time to transmit T_i via the cellular interface, and the time to transmit it back via the D2D interface. Similar to energy, there are two cases to compute the delay of the first remote task. In Case 1, the delay of the first task in the remote queue can be estimated by $1/\lambda^p_l$. In Case 2, the first task of the remote queue needs to wait for a while until the total data size of the remote queue is larger than Γ, which is $\frac{\Gamma - S^p_r}{\lambda^p_r \times \overline{S^p_r}}$. The queue delay depends on the total remote task size S^p_r. The cellular network delay and the D2D delay depend on d_i. Putting them together, the delay to execute T_i at the proxy node p is:

$$D^{i,p}_{remote} = \frac{d_i + S^p_r}{r^p_{cell}} + \frac{d_i}{r_{D2D}} + max\{1/\lambda^p_l, \frac{\Gamma^p - S^p_r}{\lambda^p_r \times \overline{S^p_r}}\} \quad (5)$$

4.4.3 Proxy Selection and Congestion Avoidance

In QATO, a node periodically asks all nodes in its proxy list for the network information, including the downlink/uplink throughput, the Γ values for downlink/uplink, the local/remote task coming rate (λ_l/λ_r), the total remote task size (S_r), and the average remote task size ($\overline{S_r}$). The update interval introduces a tradeoff between obtaining accurate network topology and saving energy. The selection of this interval should consider the real context. For example, if a user is on a train between two cities, the neighbors are relatively stable and the update interval should be longer. If a user is on a bus inside a city, the neighbors change frequently and the update interval should be smaller. We let the users select the update interval based on their context. By default the update interval is set to 1 minute. Between two updates, a node will use the previous history information to estimate the load on a given proxy.

Given a list of proxy P and the network information of each proxy p, a node will run the proxy selection algorithm for task T_i as shown in Algorithm 2. For each proxy p, the node computes the energy and delay saving ratio when offloading T_i to p, and only consider the proxy that can save both energy and delay. These proxy candidates are ordered by the total saving ratio as computed by Eq. 6, where α is a weight parameter to balance the energy and delay saving during offloading. If α is 1, only energy saving is considered. If α is 0, only delay saving is considered. This parameter can be adjusted based on the users' requirement. In QATO, we set α to 0.5 to consider energy and delay equally. Given the ordered proxy candidates, the node sends requests to them one by one. If the request is accepted by p, it will be selected as the proxy. After offloading T_i to proxy p, the node updates the network information of p, including the remote task coming rate, the total remote queue size and the average remote task queue size.

Considering both energy and delay can help to avoid congestion at the proxy. If one proxy p has higher throughput, its remote

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing

Algorithm 2 Proxy Selection

Input: a list of proxies P, and the network information of each proxy p including r_{cell}^{p}, λ_l^p, λ_r^p, S_r^p, and $\overline{S_r^p}$
function PROXYSELECTION(P, T_i)
  Proxy candidate set $Candidates \leftarrow \emptyset$
  for each proxy $p \in P$ **do**
    if $E_{local}^{i} > E_{remote}^{i,p}$ and $D_{local}^{i} > E_{remote}^{i,p}$ **then**
      Compute $saving_p$ using Eq. 6
      $Candidates \leftarrow Candidates \bigcup p$
    end if
  end for
  Order $Candidates$ by the saving ratio
  $Proxy \leftarrow null$
  for each proxy $p \in Candidates$ **do**
    if p accepts the node's request **then**
      $Proxy \leftarrow p$
      Update λ_r^p, S_r^p and $\overline{S_r^p}$
      Quit the For loop
    end if
  end for
  if $Proxy = null$ **then**
    Run T_i locally
    Announce itself as a proxy
  end if
end function

queue will grow quickly. Then the queue delay for remote tasks will increase and the total saving ratio will be smaller. As a result, other remote tasks will be scheduled to other proxies. Later, the remote tasks on p are executed and its queue delay will decrease. This information will be sent to other nodes, and more remote tasks will be offloaded to p to achieve load balance.

$$saving_p = \alpha \frac{E_{local}^{i} - E_{remote}^{i,p}}{E_{local}^{i}} + (1-\alpha)\frac{D_{local}^{i} - D_{remote}^{i,p}}{D_{local}^{i}} \quad (6)$$

Proxy's Response: From the proxy's perspective, serving more users can aggregate more tasks and save more energy. However, serving more remote tasks will increase the length of the remote task queue and thus increase the average delay for remote tasks. Next, we illustrate how the number of users affect the delay. Assume a proxy serves N users. Suppose at time t, each user generates one data request (task), and all remote tasks are offloaded to the proxy. Then, the delay of all tasks by using QATO is denoted as D_{remote}. For comparison, we also compute the total delay without using QATO, which is denoted as D_{local}. The delay saving ratio is computed as $(D_{local} - D_{remote})/D_{local}$. If this ratio is negative, it indicates that the proxy serves too many users and introduces more delay.

Suppose the proxy uses Carrier 2's LTE network and the other N users use Carrier 1's HSPA+ network as shown in Table 2. Fig. 10 illustrates the delay saving ratio when the number of users changes. As can be seen, when more users offload traffic to one proxy, the delay saving ratio drops quickly since the queue delay increases. When the number of users is fixed, the delay saving ratio decreases as the average data size increases. This is because the D2D delay is becoming much larger during traffic offloading. Therefore, the proxy should also consider the average data size when determining the number of users to serve. For example, as shown in Fig. 10, one proxy can support 11 users when the average data size is 1MB, but can only support 3 users when the data size increases to 10MB. In QATO, the proxy can adaptively adjust the number of users to serve based on the average remote task size. When there are enough users, it will reject later requests.

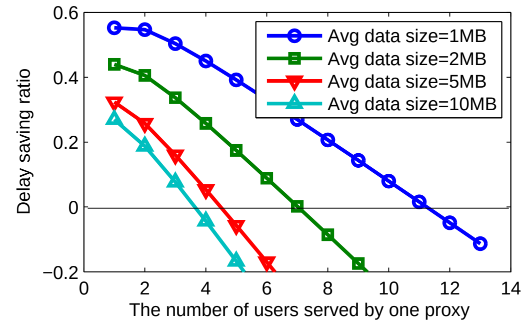


Fig. 10. The number of users a proxy can serve

TABLE 4
Relationship between energy, delay and bandwidth

Energy (J)	Time (sec)	Bandwidth (KB)
1.9	1	576

4.4.4 Proxy Failure and Recovery

The connection between a node and its proxy may be lost due to node movement. Thus, nodes should monitor the status of their proxy continuously. A node should find a new proxy if the selected proxy is unavailable, or if the delay going through the proxy is too long. To search for a new proxy, the node uses service discovery to search for the proxy list, as mentioned in Section 4.2. If there are proxy candidates, it communicates with them to find a new proxy as described in Section 4.4.3. If the proxy list is empty or there is no proper proxy candidate, the node will use its own cellular interface and announce itself as a proxy in the next round of information update.

4.5 Credit Manager

We design a credit-based scheme to motivate nodes to help others. The credit is a kind of virtual money used to measure the cost of the proxy. As time, energy, and bandwidth are all valuable resources, we consider all of them in terms of credits. Since these resources have different unit, we introduce the idea of equivalents to compare their value. For example, Carrier 1's HSPA+ network takes 1 second to download 4.5M bit (576K Byte), and consumes 1.9 Joule of energy, as shown in Table 4, and thus we consider these costs at the same value. To be more general, we introduce three parameters ρ_e, ρ_d and ρ_b, to adjust the unit and combine energy, delay and bandwidth together, so that $\rho_e Energy = \rho_d Delay = \rho_b Bandwidth$. These three parameters may vary due to the network quality, and the credit manager can adjust them in real time. The credit can also be extended to exchange with real money, but this is out of the scope of this paper.

If task T_i is executed locally, a node will cost energy, delay and bandwidth. By offloading T_i, the node can save such cost, which is denoted as C_{save}^{i}. This is the maximum credit a node wants to pay to the proxy when offloading T_i. On the other hand, the proxy has to pay extra energy and bandwidth to get T_i, and the total cost is C_{cost}^{i}. A proxy should get at least C_{cost}^{i} credit back when helping others to offload T_i.

$$C_{save}^{i} = \rho_e \times E_{local}^{i} + \rho_d \times (D_{local}^{i} - D_{remote}^{i}) + \rho_b \times d_i \quad (7)$$

$$C_{cost}^{i} = \rho_e \times E_{remote}^{i} + \rho_b \times d_i \quad (8)$$

As task T_i is offloaded only when it saves both energy and delay, we have $C_{save}^{i} > C_{cost}^{i}$. To motivate both users to use QATO,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing

some credits in-between should be paid to the proxy as shown in Equation 9, where β is a parameter to balance the cost of proxy and normal nodes, and it is set to 0.5 in our system. Note that other incentive schemes like [36] can also be applied.

$$C^i_{paid} = C^i_{cost} + \beta(C^i_{save} - C^i_{cost}) \tag{9}$$

When proxies are used, there will be security and privacy issues since the proxy can be a malicious attacker. This is similar to the case of using a public access point, where the access point has similar functions to the proxy, and then similar techniques can be applied. Note that many applications may already have security mechanisms to address this problem. For example, many banking applications are based on HTTPS where end-to-end security is applied, and thus security is not a problem. Other solutions can be found in [20]. Since this is not the focus of this paper, we will not further discuss it.

5 QATO Implementation

QATO is implemented on the Android platform. In this section we introduce the implementation of QATO and two applications designed based on QATO.

5.1 Implementation Details

The four components of QATO, service discovery, offload engine, data transmission and credit manager, are shown in Fig. 6, where each component is implemented with a thread, and message passing is used for communication.

In the service discovery module, each device registers “QATO” as a service in the local network as mentioned early. Since the service name should be unique in a local network, the Android system automatically adjusts the service name on a device to a format like “QATO(1)” when there is conflict. Therefore, during the service discovery, all service names containing “QATO” are treated as the same service.

In the offload engine, a major task is to measure the network status such as throughput. Throughput measurement has two challenges. First, it introduces extra bandwidth and energy cost. Second, when a user moves and the cellular signal strength changes, the previous measurement will be inaccurate. To address these problems, we leverage the relationship between throughput and signal strength, and measure the throughput when users are transmitting useful data. In Android, a device can retrieve the signal strength using `getGsmSignalStrength()`. This call has no extra cost since the device measures the wireless signal strength by default. The result of this call is an integer ranging between 0 to 31 when the signal is valid, where a larger number indicates stronger signal, as defined in 3GPP TS 27.007 [1]. If the signal is not known or not detected, 99 is used. We maintain a table with the signal strength as the key and the downlink/uplink throughput as the value. When a node transmits data, the signal strength is recorded and the downlink/uplink throughput is updated. With this table, throughput information can be obtained based on the recent signal strength. Although this method may not always get the accurate throughput, it can be used to estimate the throughput due to its low cost. Also, the throughput estimation will be updated after each data transmission. Thus, even if the throughput estimation is not accurate, it only affects the first task. To accurately estimate the throughput, other existing solutions [19] [32] can be used, although they may have higher cost.

```
// original interface
Socket socket = new Socket (serverIP, serverPort);
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
......
// QATO interface
QATOSocket socket = new QATOSocket (serverIP, serverPort,
                                    proxyIP, proxyPort);
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
......
```

Fig. 11. QATO TCP interface

In the data transmission module, to schedule a remote task, we need to know whether the network interface is in the data transmission state as mentioned in Section 4.3. However, this information is not available in our testbed (Android 4.2.2). One solution is to monitor the arrival time of all local and remote tasks. The arrival time of remote tasks can be easily monitored as they are offloaded by QATO, but the arrival time of local tasks can only be obtained when the tasks are generated by apps under our control (e.g., Web Browser and Photo Uploader). The problem can be solved after Android 5.0, where the system provides a callback interface `OnNetworkActiveListener`, which can tell whether the network interface is transmitting data. Thus, QATO should perform better in Android 5.0 or later version.

5.2 QATO Interface

All components of QATO are wrapped into classes and run in the background when QATO is turned on. For developers, QATO provides a simple interface. Using TCP as an example (UDP works similarly), the QATO interface is shown in Fig. 11. Originally a client connects to the server directly. After successful connection, a client can manipulate the input/outputstream. In QATO, we provide a new QATOsocket with two more parameters: proxy IP address and port.

All network tasks using the QATOsocket will be forwarded to QATO. The offload engine decides whether to offload the tasks. If not, the connection works the same as the original one. Otherwise, the node first creates a connection to the proxy via the WiFi direct interface, and then the proxy creates a new connection to the real server. There will be two connections on the proxy simultaneously. Later on, the proxy connects the input/outputstream of one connection to the output/inputstream of the other one. In this way, the proxy works like a tunnel to transmit data, without storing the user data, and as a result the user privacy is also protected. For developers, the whole process is transparent and they can use the inputstream and outputstream as normal.

5.3 Applications

We have developed two applications on top of QATO: a web browser focusing on download offloading, and a photo uploader focusing on upload offloading.

5.3.1 *Web Browser*

In the web browser application, the opening of a webpage is treated as downloading multiple files. To eliminate the effect of congestion on the web server, all files (including embedded objects) of the webpage are downloaded to a server in our lab. When the phone opens a webpage, we use the idea in [35] by downloading all object files first and then rendering them. It works as below. After a user enters a URL and clicks the “go” button,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing

TABLE 5
Webpage benchmarks

Name	URL	# of Files	Size (KB)
Google	www.google.com	2	10
Yahoo	www.yahoo.com	165	808
Amazon	www.amazon.com	9	126
Wiki	www.wikipedia.org	18	117
Ebay	www.ebay.com	13	221
Bing	www.bing.com	2	29
Craigslist	www.craigslist.com	7	409
Go	www.go.com	18	980
Espn	www.espn.go.com	53	598
CNN	www.cnn.com	21	396

TABLE 6
Photo benchmarks

ID	Data Size (KB)	ID	Data Size (KB)
1	38	11	1028
2	55	12	1092
3	83	13	1171
4	309	14	1255
5	382	15	1280
6	392	16	1293
7	393	17	1297
8	519	18	1479
9	678	19	2539
10	817	20	2882

the web browser downloads the main webpage. Then it parses the whole content, detects all embedded object files, including CSS, images, javascript files, and downloads them together. After all files are downloaded, the web browser modifies the object links in the main webpage and redirect them to the local files. Then the webpage can be displayed. By downloading all webpage files in a bunch, the tail energy can be significantly reduced.

5.3.2 Photo Uploader

With cameras on smartphones, users are generating more and more photos, and uploading them to Facebook, flickr, google, etc. Since photos are very large, it may take much more time to upload photos and consume a large amount of energy when the wireless signal is not good. Thus, it is better to offload such tasks to neighboring nodes with better network quality. To achieve this goal, we design a photo uploader based on QATO. Users can take photos or select photos from the gallery and then upload them to a self defined server. Since photo uploading is delay tolerant, the proxy can adjust Γ to a balance between saving energy and reducing delay.

6 Performance Evaluations

In this section, we first run some experiments to show the efficiency of QATO, and then use trace driven simulations to show that proxy nodes can also get benefits in the long run and QATO can realize load balance among multiple proxies. We compare the performance of QATO, denoted as "`Ours`", to the original method (without traffic offloading), denoted as "`Original`".

6.1 Real Experiments

We have implemented QATO on two smartphones as listed in Table 1 and run experiments in City 1. All phones have Android 4.2.2 and have pre-installed two applications: web browser and photo uploader. For the `Original` method, we run each application using the GS3 phone individually. For `Ours` method, we turn on QATO on both phones and put them within communication range. As LTE has larger downlink throughput than HSPA+, the GS4 phone is selected as the proxy.

For both methods we use Monsoon power monitor to measure the energy as described in Section 3.2. In the original method, we only consider the data transmission energy and delay on GS3. When QATO is used, we also consider the energy on both client and the proxy (e.g., the GS4 phone in our case). For the delay, we consider both cellular and D2D transmission delay.

6.1.1 Web Browser Results

We pick 10 most popular websites from the Alexa website [2], as listed in Table 5 to test the performance of download offloading. These websites have different numbers of embedded objects. Some contain one image while others contain hundreds of images. In this application, the delay is defined as the time from a user pressing the "go" button to the time when the webpage is totally downloaded. Since web browser is not delay tolerant, we set Γ to the minimum webpage size so that all offloading requests can be executed immediately.

Figure 12 compares the energy and delay of the two methods. When downloading webpages, the original method takes more energy and time than ours since QATO can offload traffic to the GS4 phone, which has much higher downlink throughput. On average, our method can save energy by 38% and reduce delay by 45%.

6.1.2 Photo Uploader Results

We use 20 photos with different sizes to evaluate the performance of upload offloading. The data size is listed in Table 6, which ranges from several kilobytes to several megabytes, with an average size of 939KB. The photos are roughly divided into two categories: small photos with data size smaller than 1MB, and large photos larger than 1MB.

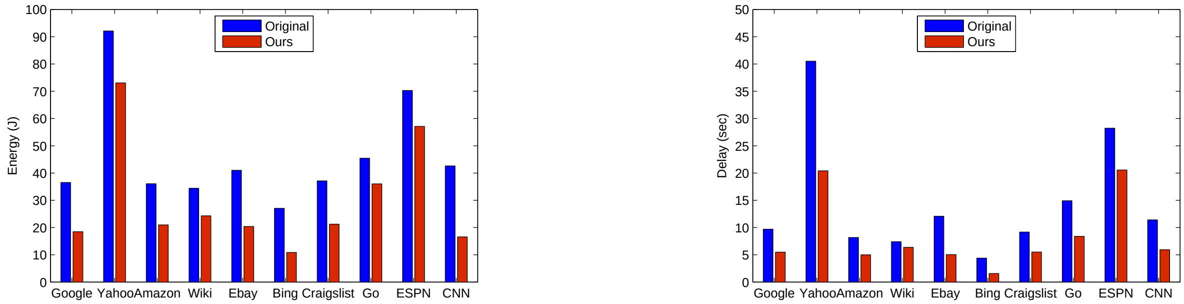
The comparison results of the original method and our method are shown in Fig. 13. The dotted line shows the average value of small photos and large photos using the original method. In the original method, the energy and delay of the small photos are much smaller than that of the large photos. More specifically, the average energy of the small photos is $23.4J$, while it is $54.1J$ for large photos. The average delay of small photos is 3.57 seconds while it is 17.63 seconds for large photos. This is because the uplink bandwidth of HSPA+ is relatively small. However, when using QATO, the energy and the delay are both reduced significantly, since the GS4 phone has much higher uplink bandwidth. For all photos, our method can save 70% of energy and 88% of delay on average.

The data size threshold Γ affects the performance of traffic offloading. Since photo uploading is delay tolerant, adding more delays (i.e., increasing Γ) can be used to save more energy. Suppose a user selects 20 photos to upload and the duration of selecting one photo is 5 seconds, the energy and delay of using QATO with different Γ are shown in Fig. 14. When Γ increases, more energy is saved since more data can be aggregated to transmit at once and more tail energy is saved. However, the delay also increases since there will be more queue delay and promotion delay.

6.1.3 Energy Consumption of Different Components

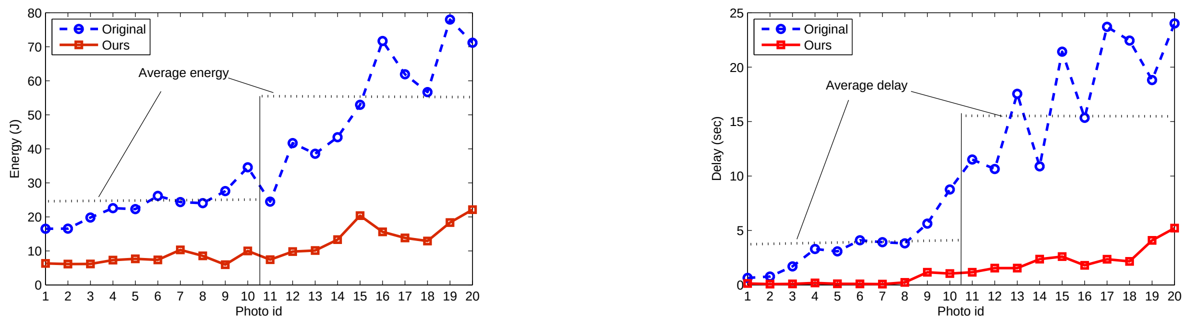
In this subsection, we evaluation the energy consumption of different components in QATO, which includes the cellular interface, D2D, service discovery, and system maintenance such as

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing



(a) Energy (b) Delay

Fig. 12. Energy and delay comparisons with/without QATO when downloading webpages.



(a) Energy (b) Delay

Fig. 13. Energy and delay comparisons with/without QATO when uploading photos.

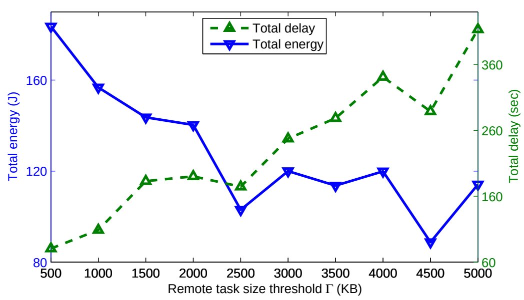


Fig. 14. The impact of Γ on the performance of photo uploading

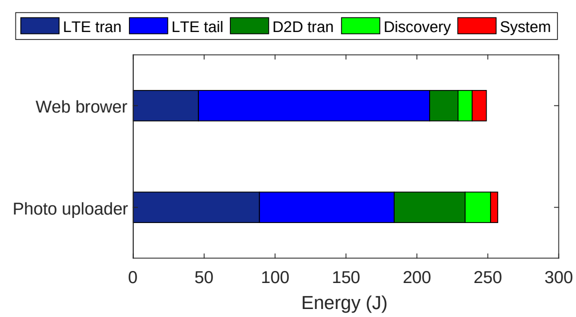


Fig. 15. Energy consumption of different components of QATO

measuring throughput and scheduling data transmissions. In the measurement, we carefully analyzed the power consumption trace and the data transmission trace to extract the data transmission energy, and then run the service discovery and system maintenance separately to obtain their energy consumption. Fig. 15 shows the energy consumption of different components for web browsing and photo uploading. Generally speaking, data transmission (cellular, D2D) consumes most of the energy, among which the cellular interface consumes much more energy than the D2D interface. Service discovery is executed periodically, and the energy for system maintenance is mainly affected by the number of tasks to be scheduled.

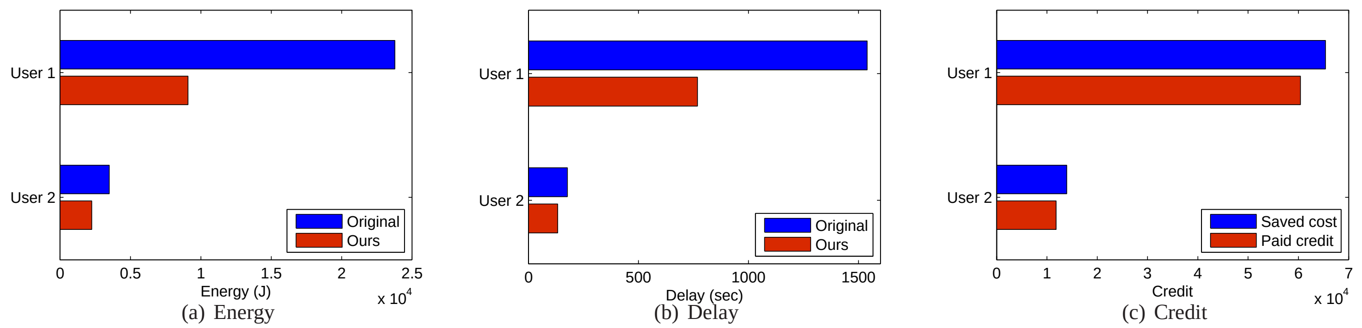
For data transmission, photo uploading consumes much more energy (both LTE and D2D) than web browsing, since it has much larger data size. Web browsing consumes much more tail energy (LTE tail) due to the following reason. Web pages contain multiple files which are downloaded separately, and users read a web page for some time before opening another one. Thus, there exists many idle time periods between downloading, introducing more tail energy. On the other hand, photos are uploaded in several batches, with less idle time interval between data transmissions.

6.2 Trace-driven Simulations

In this section, we aim to show that all nodes, including the proxy nodes, can get benefit in the long run. We collect network traces from 4 users in one month. Then we assume two commuters, User 1 with Carrier 1's data plan and User 2 with Carrier 2's data plan, always travel between City 1 and City 2. We feed trace 1 and trace 2 to User 1 in City 1 and City 2, respectively. Similarly trace 3 and trace 4 are fed to User 2 in two cities. As shown in Table 2, User 1 offloads traffic to User 2 in City 1 and User 2 offloads traffic to User 1 in City 2. In these two month periods, we compare the performance with/without QATO and the results are shown in Fig. 16.

Fig. 16(a) and Fig. 16(b) compare the energy and delay of the original method and QATO. As can be seen, both users benefit from using QATO by saving energy and reducing delay. User 1 consumes much more energy and time than User 2 in the original method because he has more network tasks. Due to the large

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing



(a) Energy (b) Delay (c) Credit

Fig. 16. Performance comparisons with/without traffic offloading

TABLE 7
Energy and delay comparison of different methods

	Energy (J)	Delay (sec)
`Original`	1309.4	217.6
`Single-proxy`	163.9	162.7
`CarrierMix`	223.6	103.9
`Ours`	189.3	43.9

number of tasks, User 1 has more opportunity to save energy and reduce delay by offloading them to User 2. In total User 1 saves 62% of energy while User 2 saves 36%, and User 1 reduces delay by 50% while User 2 reduces the delay by 25%.

The credit (cost) value considers energy, delay and bandwidth. As mentioned before, when offloading a task, a node saves some cost which is more than the credits paid to the proxy. From this point of view, we say a node benefits from QATO if the saved cost is more than the paid credits in the long run. To verify this assumption, we compare the total saved cost and the paid credits during the two month periods for both users and show the results in Fig. 16(c). It clearly verifies our assumption since both users paid less credits than their saved cost. The benefit (the difference between saved cost and paid credit) of User 1 is larger since User 1 offloads more tasks. On the other hand, User 2 also earns some credits (which is not shown in the figure) by being a proxy.

6.3 Synthetic Trace-Driven Simulations

In this subsection, we use synthetic trace with more users to demonstrate that QATO can save more energy with more users and achieve better load balance among proxies. In the simulation, we assume that the throughput of the proxy follows normal distribution $N(15.4Mbps, 5.5Mbps)$, and the throughput of the client users have lower throughput following distribution $N(4.5Mbps, 1.3Mbps)$. For each user, task arrival follows a Poisson distribution with an average interval of 60 seconds, and the data size of each task is randomly distributed from 10KB to 3MB.

6.3.1 Impact of the Number of Users on Energy Saving

If there are many users in one area, they will form several groups as discussed in Section 4.4.3. In each group, a smaller number of users with high throughput will serve as proxies, and other users with low throughput will work as normal clients. In this subsection, we study the impact of the number of users on energy consumption. Fig. 17 shows the average energy saving ratio of QATO as a function of the number of users, where the number of users is increased from 2 to 10, with at least one and at most 30% of the users serving as proxies. The test was run 10 times and 10 minutes each time. As can be seen, the energy saving ratio increases from 48% to 85% when the number of users increases from 2 to 10. This is because more tail energy will be reduced when more tasks are aggregated. When the number of users further increases (near 10), the energy saving ratio will not further increase, but the delay may become longer as shown previously in Fig. 10. Thus we suggest not to assign too many users to one group.

6.3.2 Load Balancing at the Proxies

When there are multiple proxies, load balancing is necessary for achieving a balance between saving energy and reducing delay. In this section, we compare QATO with others in terms of energy saving and delay. Besides the `Original` and `Ours` methods, we add another method called `Single-proxy` which schedules all tasks to the proxy with the highest throughput. We also compare the performance with the threshold based offloading method in [21] (referred as `CarrierMix`). CarrierMix computes the energy (both cellular and D2D energy) consumed at each node periodically and the node with the minimum energy cost is the proxy.

In the simulation, we assume there are 10 users in a group, and three of them have higher throughput. To simplify the presentation, users are ordered by their throughput, so that User 1 has the highest throughput. The energy consumption and delay of the four methods are listed in Table 7. As can be seen, all methods using proxy can save energy and reduce delay, since the proxy (proxies) has higher throughput. For energy, our solution saves 85% of energy and CarrierMix saves 83% of energy than the original solution. The most energy efficient solution is Single-proxy, which saves 87% of energy, since it only uses the proxy with the highest throughput. Moreover, it aggregates all traffic and thus has more opportunities to cut the tail energy. However, this is at the cost of delay. We can see that the single-proxy method has much longer delay than others.

To understand the traffic load at each proxy, we compare the energy consumption of the proxies, and the result is shown in Fig. 18. In the single-proxy method, User 1 works as the proxy, and it consumes the highest amount of energy. Other users are treated as client nodes and only consume a little D2D energy. CarrierMix has better energy balance among proxies, but it also selects three nodes with low throughput as proxies at some time slots. Since it uses some nodes with low throughput as proxies, it has more energy consumption and longer delay than our method (delay is shown in Table 7). QATO uses the first three nodes as proxies to avoid congestion, and the three proxies have similar energy consumption. The proxy of User 1 handles more tasks and thus consumes more energy. Compared to other methods, QATO

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2017.2690296, IEEE Transactions on Mobile Computing

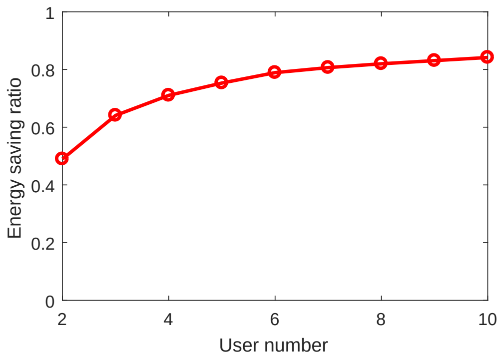


Fig. 17. The energy saving ratio with different number of users in a group

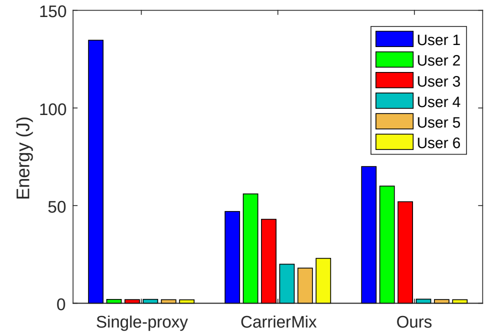


Fig. 18. The load balance of our scheduling algorithm

can achieve better load balance among proxies with low energy consumption and low delay.

7 Related Work

Our work aims to reduce energy and delay by offloading traffic to neighbors with better service quality. It is related to three categories of work.

Power saving in cellular networks: In cellular networks, including 3G, 4G and LTE, the radio interface on smartphone is kept in the high power state (tail state) for a long time after data transmission, which may waste a large amount of energy. To solve this problem, some researchers introduce methods to aggregate the network traffic to amortize the tail energy [12, 13], or turn the radio interface off quickly by predicting the end of communication [7, 27].

Quality aware data access: The service quality difference of cellular network within an area has drawn researchers' attention. The Bartendr project [29] works on the user side and suggest a user to defer data transfer until reaching a location with better signal. Another approach, coordinated multipoint transmission/reception (CoMP) technology [28, 16], works on the network side and leverages the coordination between multiple BSs to improve the throughput. Different from them, we leverage users' cooperation to save energy and reduce delay.

Offloading: As the cellular network is crowded in some locations, lots of research has been done to offload cellular traffic to WiFi networks to reduce the traffic [22] and increase the network throughput [31]. Since WiFi is not always available, researchers also propose to offload cellular traffic to D2D networks, such as Bluetooth and WiFi direct [11, 10, 17, 5]. Among all D2D interfaces WiFi direct [6] attracts more attention since it has much higher throughput. It has also been used in the standard of proximity service (ProSe) in 3GPP [4, 24]. However, the previous work of traffic offloading is mainly done via analysis and simulations [23], and there is few real implementation [6].

Besides reducing the load of cellular networks, traffic offloading is also possible to leverage neighboring nodes with good signal strength, such as UCAN, which offloads (relays) data to nodes with higher throughput via the 802.11 interface [25]. However, it only analyzes the benefit based on simulations. Different from it, our work is a full implementation and runs on real devices. We also consider many practical scheduling issues related to the long tail problem, which are not considered in UCAN. CarrierMix is another work that uses the similar idea to offload traffics between users under different cellular carriers [21]. They introduce two online traffic offloading methods: max rate and threshold based offloading. The previous one offloads traffics to one proxy with the maximum throughput, which does not consider load balance. The later one considers load balance from the perspective of energy, but it increases the delay in some cases.

There are also works on mobile clouds [8, 18, 30] which aim to offload complex computations to cloud to save energy. Recently, many researchers also consider offloading computations to nearby mobile devices [9] to save energy. Their idea of leveraging neighbors' resource inspires our work, but their works focus on computation offloading whereas our work focuses on communication offloading.

8 Conclusions

In this paper, based on real measurements, we demonstrated the existence of significant service quality difference between wireless carriers at the same locations, and then motivated the necessity of node collaboration to save energy and reduce delay in cellular networks. We proposed a traffic offloading framework QATO to offload network tasks to neighboring nodes with better service quality, so as to save energy and reduce delay. QATO can find neighbors through service discovery without the support of infrastructure network and offload traffic to neighbors with higher throughput considering both energy and delay. QATO also provides incentive mechanisms to motivate nodes to help each other. To validate our design, we have implemented QATO on Android platform and developed a web browser and a photo uploader on top of it. Experimental results show that QATO can reduce energy by 38% in downloading and 70% in uploading, and reduce delay by 45% in downloading and 88% in uploading. Through trace-driven simulations, we also show that all users can benefit from data offloading in the long run and QATO can realize load balance among proxies.

References

[1] 3gpp ts 27.007. http://m10.home.xs4all.nl/mac/downloads/3GPP-27007-630.pdf.

[2] Alexa top sites. http://www.alexa.com/topsites.

[3] Shannon-hartley theorem. http://en.wikipedia.org/wiki/Shannon-Hartley_theorem.

[4] 3rd generation partnership project; technical specification group sa; fesibility study for proximity service (prose) (release 12). In *TR 22.803 V1.0.0*, 2012.

[5] S. Andreev, A. Pyattaev, K. Johnsson, O. Galinina, and Y. Koucheryavy. Cellular traffic offloading onto network-assisted device-to-device connections. *Communications Magazine, IEEE*, 52(4):20–31, April 2014.

[6] A. Asadi, Qing Wang, and V. Mancuso. A Survey on Device-to-Device Communication in Cellular Networks. *IEEE Communications Surveys Tutorials*, 16(4):1801–1819, 2014.

[7] Pavan K. Athivarapu, Ranjita Bhagwan, Saikat Guha, Vishnu Navda, Ramachandran Ramjee, Dushyant Arora, Venkat N. Padmanabhan, and

George Varghese. RadioJockey: Mining Program Execution to Optimize Cellular Radio Usage. In *ACM MobiCom*, 2012.

[8] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *ACM MobiSys*, 2010.

[9] Yeli Geng, Wenjie Hu, Yi Yang, Wei Gao, and Guohong Cao. Energy-Efficient Computation Offloading in Cellular Networks. In *IEEE ICNP*, 2015.

[10] Bo Han, Pan Hui, V.S. Anil Kumar, Madhav V. Marathe, Guanhong Pei, and Aravind Srinivasan. Cellular traffic offloading through opportunistic communications: a case study. In *Proceedings of the 5th ACM workshop on Challenged networks*, CHANTS '10, 2010.

[11] Bo Han, Pan Hui, V.S.A. Kumar, M.V. Marathe, Jianhua Shao, and A. Srinivasan. Mobile data offloading through opportunistic communications and social participation. *IEEE Transactions on Mobile Computing*, 11(5):821–834, 2012.

[12] Wenjie Hu and Guohong Cao. Energy Optimization Through Traffic Aggregation in Wireless Networks. In *IEEE INFOCOM*, 2014.

[13] Wenjie Hu and Guohong Cao. Quality-Aware Traffic Offloading in Wireless Networks. In *ACM MobiHoc*, 2014.

[14] Wenjie Hu, Guohong Cao, Srikanth V. Krishanamurthy, and Prasant Mohapatra. Mobility-Assisted Energy-Aware User Contact Detection in Mobile Social Networks. In *IEEE ICDCS*, 2013.

[15] Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *ACM MobiSys*, 2012.

[16] R. Irmer, H. Droste, P. Marsch, M. Grieger, G. Fettweis, S. Brueck, H.-P. Mayer, L. Thiele, and V. Jungnickel. Coordinated multipoint: Concepts, performance, and field trial results. *Communications Magazine, IEEE*, 49(2):102–111, 2011.

[17] Lorenzo Keller, Anh Le, Blerim Cici, Hulya Seferoglu, Christina Fragouli, and Athina Markopoulou. MicroCast: Cooperative Video Streaming on Smartphones. In *ACM MobiSys*, 2012.

[18] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. ThinkAir: Dynamic Resource Allocation and Parallel Execution in the Cloud for Mobile Code Offloading. In *IEEE INFOCOM*, 2012.

[19] Swarun Kumar, Ezzeldin Hamed, Dina Katabi, and Li Erran Li. LTE Radio Analytics Made Easy and Accessible. In *ACM SIGCOMM*, pages 211–222, 2014.

[20] M. La Polla, F. Martinelli, and D. Sgandurra. A Survey on Security for Mobile Devices. *IEEE Communications Surveys Tutorials*, 15(1):446–471, 2013.

[21] J. Lee, K. Lee, Y. Kim, and S. Chong. CarrierMix: How Much Can User-side Carrier Mixing Help? *IEEE Transactions on Mobile Computing*, 16(1):16–29, 2017.

[22] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile Data Offloading: How Much Can WiFi Deliver? *IEEE/ACM Transactions on Networking*, 21(2):536–550, 2013.

[23] N. Lee, X. Lin, J. G. Andrews, and R. W. Heath. Power control for d2d underlaid cellular networks: Modeling, algorithms, and analysis. *IEEE Journal on Selected Areas in Communications*, 2015.

[24] Xingqin Lin, J. Andrews, A. Ghosh, and R. Ratasuk. An Overview of 3GPP Device-to-Device Proximity Services. *IEEE Communications Magazine*, 52(4):40–48, 2014.

[25] Haiyun Luo, Ramachandran Ramjee, Prasun Sinha, Li (Erran) Li, and Songwu Lu. UCAN: A Unified Cellular and Ad-hoc Network Architecture. In *ACM MobiCom*, 2003.

[26] Chunyi Peng, Suk-Bok Lee, Songwu Lu, Haiyun Luo, and Hewu Li. Traffic-driven Power Saving in Operational 3G Cellular Networks. In *ACM MobiCom*, 2011.

[27] Feng Qian, Zhaoguang Wang, A. Gerber, Z.M. Mao, S. Sen, and O. Spatscheck. TOP: Tail Optimization Protocol for Cellular Radio Resource Allocation. In *IEEE ICNP*, 2010.

[28] M. Sawahashi, Y. Kishiyama, A. Morimoto, D. Nishikawa, and M. Tanno. Coordinated multipoint transmission/reception techniques for LTE-advanced [Coordinated and Distributed MIMO]. *Wireless Communications, IEEE*, 17(3):26–34, June 2010.

[29] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Bartendr: A Practical Approach to Energy-aware Cellular Data Scheduling. In *ACM MobiCom*, 2010.

[30] Cong Shi, Karim Habak, Pranesh Pandurangan, Mostafa Ammar, Mayur Naik, and Ellen Zegura. COSMOS: Computation Offloading As a Service for Mobile Devices. In *ACM MobiHoc*, 2014.

[31] Hamed Soroush, Peter Gilbert, Nilanjan Banerjee, Mark D. Corner, Brian N. Levine, and Landon Cox. Spider: Improving Mobile Networking with Concurrent Wi-Fi Connections. In *ACM SIGCOMM*, 2011.

[32] Xiufeng Xie, Xinyu Zhang, Swarun Kumar, and Li Erran Li. piStream: Physical Layer Informed Adaptive Video Streaming over LTE. In *ACM MobiCom*, pages 413–425, 2015.

[33] SeungJune Yi, SungDuck Chun, YoungDae Lee, SungJun Park, and SungHoon Jung. *Radio Protocols for LTE and LTE-Advanced*. Wiley, 2012.

[34] Jiansong Zhang, K. Tan, Jun Zhao, Haitao Wu, and Yongguang Zhang. A Practical SNR-Guided Rate Adaptation. In *IEEE INFOCOM*, 2008.

[35] Bo Zhao, Wenjie Hu, Qiang Zheng, and Guohong Cao. Energy-Aware Web Browsing on Smartphones. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 26(3):761–774, 2015.

[36] Xuejun Zhuo, Wei Gao, Guohong Cao, and Sha Hua. An Incentive Framework for Cellular Traffic Offloading. *IEEE Transactions on Mobile Computing*, 13(3):541–555, 2014.

Wenjie Hu received the B.S. degree in computer science from Tongji University in 2007, the M.S. degree in Computer Science from Tsinghua University in 2010. He is a PhD Candidate at the department of Computer Science and Engineering in the Pennsylvania State University. His research interests include energy management for smartphones, mobile cloud and mobile video. He is a student member of the IEEE.

Guohong Cao received the BS degree in computer science from Xian Jiaotong University and received the PhD degree in computer science from the Ohio State University in 1999. Since then, he has been with the Department of Computer Science and Engineering at the Pennsylvania State University, where he is currently a Professor. He has published more than 200 papers in the areas of wireless networks, mobile systems, Internet of things, security and privacy, which have been cited more than 17000 times. He has served on the editorial board of IEEE Transactions on Mobile Computing, IEEE Transactions on Wireless Communications, IEEE Transactions on Vehicular Technology, and has served on the organizing and technical program committees of many conferences, including the TPC Chair/Co-Chair of IEEE SRDS'2009, MASS'2010, and INFOCOM'2013. He was a recipient of the NSF CAREER award in 2001. He is a Fellow of the IEEE.