Selective Traffic Offloading On the Fly: a Machine Learning Approach

Zaiyang Tang*, Peng Li†, Song Guo‡, Xiaofei Liao*, Hai Jin*, and Daqing Zhang§
*Services Computing Technology and System Lab, Cluster and Grid Computing Lab,
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
Email: {tangzaiyang, xfliao, hjin}@hust.edu.cn
†School of Computer Science and Engineering, The University of Aizu, Japan
Email: pengli@u-aizu.ac.jp
‡Department of Computing, The Hong Kong Polytechnic University, China
Email: song.guo@polyu.edu.hk
§Institut Mines-Telecom, Telecom SudParis, France
Email: daqing.zhang@telecom-sudparis.eu

***Abstract*—It has been well recognized that network transmission constitutes a large portion of smartphone energy consumption, mainly because of the tail energy caused by cellular network interface. Traffic offloading has been proposed to reduce energy by letting a smartphone offload network traffic to its neighbors in vicinity via low-power direct connections (e.g., WiFi Direct or Bluetooth). Our experiments conducted in a realistic environment reveal that energy efficiency cannot be improved or even deteriorates without a carefully designed offloading strategy. In this paper, we propose a selective traffic offloading scheme implemented as a smartphone middleware in a software-defined fashion, which consists of a packet classifier and a traffic scheduler. Using a light-weight machine learning approach exploiting unique smartphone context information, the packet classifier identifies packets generated on the fly as offloadable or not with substantially improved efficiency and feasibility on resource limited smartphones compared to traditional approaches. Both testbed and simulation based experiments are conducted and the results show that our proposal always attains the superior performance on a number of comparison metrics.**

I. Introduction

Due to the slow progress of battery technology, energy efficiency is a major concern for smartphones with limited battery capacity. It has been shown that the portion of energy consumed by the cellular network interface can be up to 50% of total energy in modern smartphones [1]. That is mainly because the cellular network interface remains to stay in a high-power state for a while after a data transmission, leading to the so-called tail energy, which dominates energy consumption of cellular network interface [2].

In order to reduce tail energy consumption, packet batching [2], [3] has been proposed to bundle non-continuous data packets and piggyback them on a later transmission. While this approach is effective for reducing tail energy, it delays the transmission of many packets and may seriously deteriorate user experience. Alternatively, some work [4], [5] has been proposed to offload data traffic to nearby smartphones via direct connections (e.g., WiFi Direct or Bluetooth) with lower power. However, the performance of traffic offloading highly depends on the quality of offloading links, which unfortunately are with limited and unstable bandwidth in practice due to user mobility.

By carefully examining smartphone traffic traces, we find that some applications, such as video streaming and file sharing, conduct bulky data transmission in "burst", while others, like social media and chatting, periodically transmit small traffic in a "non-burst" fashion. Burst traffic usually requires low latency and high bandwidth, but non-burst traffic can tolerate a certain level of delay. For example, online chatting applications periodically send heartbeat packets to the server to report their status. They can tolerate the loss or delay of one or two heartbeat packets, without affecting user experiences. A recent study [6] on traces collected from thousands of smartphones shows that about 20% of network traffic is from background data that are delay-tolerant.

Although non-burst traffic has small payload, it consumes a large portion of energy due to the tail energy caused by non-continuous data transmission. Our measurement shows that non-burst traffic consumes more than 50% of battery energy. Similar phenomena have also been confirmed by some recent studies. Huang et al. [7] have shown that background traffic periodically generated by apps makes up 84% of total network energy consumption. According to statistical analysis of AT&T [8], the "non-burst traffic" only accounts for 0.2% of the total network payload, but consumes more than 40% of battery energy.

Motivated by smartphone traffic characteristics, we propose a selective offloading approach to reduce smartphone energy consumption while guaranteeing the quality of user experience. Its basic idea is to identify offloadable data packets, which are usually delay-tolerant and small enough to go through opportunistic offloading links, from network traffic generated by applications. These packets are cached until offloading links emerge or piggybacked on a later cellular transmission, so that tail energy can be significantly reduced. Other packets, whose offloading would not improve or even degrade energy efficiency or user experiences, are referred to as unoffloadable packets in this paper. They are transmitted

immediately via the cellular network interface once they are generated, without affecting user experience. A similar process can be applied for data downloading requests.

To implement selective offloading on smartphones, we design a smartphone middleware based on the software-defined idea. It abstracts three virtual switches that can cache and forward data between applications and different network interfaces. They are configured by a controller that makes data forwarding decisions according to traffic characteristics and offloading link qualities. To fully exploit the benefits of selective offloading, we need to address a challenge in the controller design. It is offloadable packet identification, which is affected by many factors, such as packet size, connection protocols (TCP or UDP), and quality of offloading links. When a smartphone is in a highly dynamic environment, offloading links emerge momentarily and can afford only small data transmissions, such as heartbeat packets. On the other hand, if the smartphone has a stable connection with other neighbor devices, even bulky data transmissions can be offloaded.

To address this challenge, we design a packet classifier in the controller based on a lightweight machine learning algorithm. By exploiting not only the traditional packet header information (e.g., source and destination IPs, ports, and protocol) but also some unique smartphone context information (e.g., screen on/off and acceleration), our algorithm can quickly and accurately identify packets as offloadable or unoffloadable.

The main contributions of this paper are summarized as follows.

- We design a software-defined middleware that selectively offloads packets to nearby devices. This middleware separates the control logic from data caching and forwarding functions, so that it can be easily extended to implement different offloading strategies.
- To accurately identify offloadable packets, we propose an on-the-fly packet identification algorithm based on a machine learning approach. The cost-sensitive random forest algorithm for offloadable packet identification enhances its original version significantly in terms of reducing the rate of falsely identifying unoffloadable packets as offloadable ones. Such false identification would seriously deteriorate energy efficiency and user experience.
- We construct a testbed consisting of 5 smartphones, and evaluate the performance of our proposals in typical daily usage cases. Large-scale simulations on a customized simulator are also conducted to evaluate the performance of the online algorithm. Results show that our proposal outperforms existing offloading scheme by saving about 25% energy and reducing more than 50% transmission delay.

The rest of this paper is organized as follows. Section II reviews some recent work highly related to this paper. The background and motivation are presented in Section III. Section IV introduces our system design. The design of packet classifier and traffic scheduler is given in Section V. Section VI evaluates the performance of our proposals. Section VII concludes this paper.

II. Related Work

Smartphone energy drain has been studied and characterized by many recent measurement experiments. A fine-grained energy profiler called eprof has been developed in [9] and applied to measure the energy dissipation of six popular smartphone applications. The results show that the major energy consumption is on cellular and WiFi network interfaces. Ding et al. [6] have analyzed traces collected on 3785 smartphones for at least one month, and developed a power model for WiFi and 3G by incorporating signal strength factor. Chen et al. [10] have studied 800 applications running on 1520 smartphones, and their measurement results show that background energy accounts for more than 50% of total energy. A more surprising observation is that tail energy exceeds 80% of 3G/LTE energy consumption.

After recognizing the significance of energy efficiency of network transmission on smartphones, many efforts have been made to develop power saving techniques. A protocol called TailEnder [2] has been developed to reduce energy consumption of common mobile applications by postponing the transmission of delay-tolerant packets. It also prefetches packets for some web applications to improve response time and energy efficiency. Zhang et al. [3] have proposed TailTheft, a scheme that leverages the tail time for batching and prefetching to reduce energy consumption. Lei et al. [11] have proposed a *dynamic-programming* (DP) based algorithm to minimize energy consumption of a sequence of download/upload requests. Zhang et al. [12] have proposed eTrain, a transmission management system that takes advantage of IM heartbeats to piggyback aggregated delay-tolerant data.

Another approach is to exploit the cooperation among multiple smartphones for tail energy reduction. Cool-Tether [13] has been proposed to build a WiFi hotspot on-the-fly so that smartphones in the vicinity can offload their traffic via low-power WiFi connection to reduce tail energy. Hu et al. [4] have proposed a quality-aware traffic offloading framework to offload network tasks to neighboring nodes. It can identify neighbors with better service quality and motivate nodes to help each other by providing incentive mechanisms. A traffic aggregation problem has been studied in [5] to reduce the tail energy by aggregating the data traffic of multiple nodes using their P2P interfaces. An optimal offline algorithm has been designed given future task information and a heuristic algorithm is proposed and evaluated on Android smartphones for online cases. Our paper is different from above work in two aspects. First, instead of assuming stable connections to neighboring devices, we consider a dynamic environment where offloading links emerge instantaneously due to user mobility. Second, existing work offloads all data to neighboring nodes when offloading links are established, but we selectively offload the small-sized and delay-tolerant ones.

III. Preliminary and Motivation

A typical cellular network interface has three states [14]: IDLE, DCH, and FACH. The IDLE state indicates that network interface is in a sleeping mode with little power consumption.

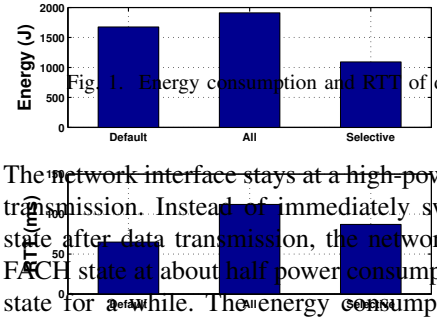

Fig. 1. Energy consumption and RTT of different approaches

The network interface stays at a high-power DCH state for data transmission. Instead of immediately switching to the IDLE state after data transmission, the network interface enters the FACH state at about half power consumption level of the DCH state for a while. The energy consumption under the FACH state is also called tail energy, which is the main cause of low energy efficiency of exiting cellular network interfaces [15].

We conduct some experiments to show the motivation of our selective offloading strategy. We run the same set of applications (including Baidu search, Sina weibo, and sohuTV) on a smartphone using different offloading approaches, and measure their energy consumption and average round-trip time (RTT) that is highly related to user experiences. The first approach is the default one (denoted by "Default") without traffic offloading, i.e., it uses only cellular network interface for data transmission. As shown in Fig. 1, it consumes about 1674J energy and the average RTT is about 65ms. Then, we consider the traditional offloading approach (denoted by "All") that traffic is forwarded to offloading links when they are available, e.g., Bluetooth links for offloading appear from time to time. It is surprising that this approach incurs even higher energy consumption than the default approach without offloading. That is because some long TCP flows would go through different paths due to interface switching, and out-of-order packets lead to retransmissions that consume extra energy. Furthermore, this approach has worse RTT because of the low rate of Bluetooth links. Finally, we identify some data packets as offloadable according to their size and interval. If the cached offloadable packets have been maintained for quite a while before an offloading link emerges, they are piggybacked to the cellular transmission. This selective offloading (denoted by "Selective" in Fig. 1 has the minimum energy consumption, while its average RTT is a little bit higher than Default, but significantly lower than All.

In above experiments, we show the promises of selective offloading in improving energy efficiency while guaranteeing low RTT, given the knowledge of traffic pattern and offloading links. In practice, it is difficult to obtain this information because we cannot predict smartphone usage pattern and user mobility with high accuracy. It motivates us to design on-the-fly selective offloading that is elaborated in the next section.

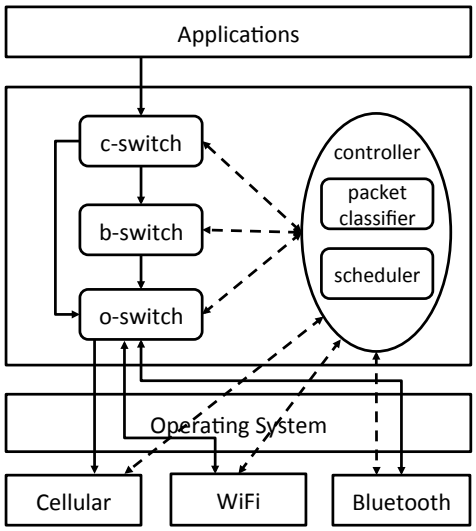

Fig. 2. System design in a software-defined fashion

IV. System Design

Our proposed selective offloading has been implemented as a middleware between the smartphone operating system and applications. It receives data packets from applications, and decides whether they are offloadable or not by using machine learning techniques according to packet header information (e.g., source and destination IPs) and smartphone status (e.g., screen on/off and moving speed). Offloadable packets are cached in a buffer, waiting for offloading links. Unoffloadable packets are transmitted immediately via cellular network interface. Meanwhile, it accepts direct connections (WiFi Direct or Bluetooth) from other smartphones, so that these offloadable data can be piggybacked on its cellular transmissions. Similarly, our middleware can handle data downloading requests issued by applications, and schedule data downloads on appropriate network interfaces.

As shown in Fig. 2, we design our middleware as a software-defined structure that separates control logic from three virtual switches along traffic flow paths from applications to network interfaces. A controller configures these virtual switches as well as network interfaces to decide transmission path of each packet. Data paths are denoted by solid arrows, and control links are shown as dashed arrows. Thanks to the software-defined structure, our middleware can be easily extended to implement other offloading strategies by deploying the corresponding controllers. In our experiments, we implement several offloading strategies based on this software-defined structure and compare them with our proposed selective offloading algorithm. The functions of main modules are elaborated as follows.

c-switch: The packet classification switch, which is also referred to as c-switch, extracts packet header information (such as packet length, protocols, source and destination IPs) of packets received from applications. All the information is sent to the controller that will decide whether packets

are offloadable or not. After receiving decisions from the controller, c-switch forwards unoffloadable packets to the o-switch that immediately transmits them over cellular network interface. Offloadable data will be forwarded to b-switch that caches them for a while.

b-switch: The b-switch has a buffer that can cache offloadable packets from c-switch. Since offloadable packets are usually with small size, b-switch will not occupy too much space. The controller will instruct the b-switch to forward these packets to the o-switch in the next hop once scheduling decisions are made.

o-switch: The offloading switch is denoted by o-switch and responsible for forwarding offloadable packets to appropriate network interfaces determined by the controller. If cellular network interface is active, receiving offloadable packets from nearby smartphones via offloading links is allowed and they will be piggybacked on the current cellular transmission. Since offloadable packets are with small size, their influence to the cellular transmission can be ignored.

Controller: The controller has two submodules. First, a packet classifier needs to decide whether packets received by c-switch are offloadable or not. Since this decision is affected by many factors, it is difficult to achieve high accuracy by using simple heuristic algorithms. We propose a lightweight machine learning algorithm based on random forest in the design of packet classifier. The controller maintains a local storage to accumulate history data for training. After training a weighted random forest, the classifier receives input that includes packet header information received from c-switch, and smartphone status, such as screen on/off, moving speed, and battery level, which is obtained by system APIs. The output is packet classification, i.e., offloadable or unoffloadable, and the corresponding forwarding decision will be sent back to the c-switch.

The second submodule is the transmission scheduler that determines how to offload packets cached in the b-switch. Although offloadable packets are usually delay-tolerant, they cannot be cached for a long time. For example, for some online chatting applications, it is tolerable to the loss or delay of one or two heartbeat packets, but not to continuous heartbeat losses which incurs poor user experience like annoying users offline indications. To avoid long waiting on the emergence of offloading links, the controller will activate the cellular network interface to transmit offloadable packets after they have been cached for a period of time.

In addition to above main modules, we design a low-energy device discovery module based on Bluetooth. Each smartphone periodically broadcasts inquiry messages, whose frequency is controlled by a scanning window. According to [16] that reveals the relationship between device discovery probability and scan frequency, we set the scanning window size as 30 seconds. Note that we choose Bluetooth instead of WiFi for device discovery because of its low energy consumption. The adaptation of WiFi and its optimization for device discovery will be included in our future work.

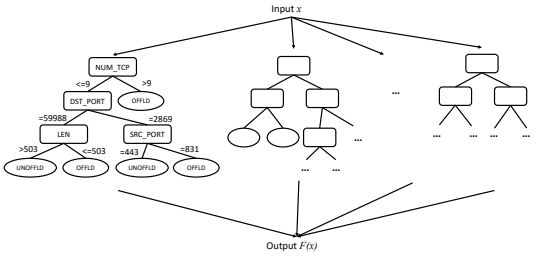

Fig. 3. The random forest

V. Identification of Offloadable Packets

Unoffloadable packets are usually generated in burst with big size. Given a traffic trace, we can identify burst traffic as unoffloadable data using the approach in [7]. Specifically, a sequence of continuous packets are identified as burst traffic if their intervals are small enough and total packet size is greater than a threshold. Although this approach is straightforward and fast, it needs future packet information that cannot be accurately predicted in practice. Furthermore, this approach ignores traffic characteristics and quality of offloading links, leading to false offloading decisions with higher energy consumption.

In this section, we propose a lightweight machine learning approach to identify offloadable packets. Two challenges are addressed in our design. First, we need to decide the information, also called attributes in this paper, required for packet identification. Instead of examining packet contents that is computationally expensive and privacy exposed, we extract packet header information (e.g., source and destination IPs, ports, packet length and protocols) that only constitutes a very low overhead. In addition, packet identification is also affected by the probability of establishing offloading links, as well as their quality if they emerge. If a mobile user has a stable WiFi Direct connection to its neighbor, even a group of burst packets can be offloadable. In contrast, only a few packets with small size could be offloaded via low-quality offloading links in a highly dynamic environment. Furthermore, we consider smartphone status, such as screen on/off, acceleration, and applications, which can be acquired by using system APIs.

The second challenge is to design a feasible machine learning algorithm that can be used on a hand-held device. Machine learning techniques have been explored in traffic classification and prediction [17] [18]. Most studies are based on a single-classifier system using only one learning algorithm, such as neural network [19] or support vector machine [20]. However, these algorithms highly depend on parameter tuning over a large amount of data to achieve high accuracy. Furthermore, since these algorithms are compute-intensive, they cannot be applied in smartphones with limited hardware resources.

We propose an algorithm, called CSRF (Cost-Sensitive Random Forest), based on random forest with high accuracy and low complexity. In the algorithm design, we need to deal with an additional challenge that the traditional random forest algorithm performs poorly in false prediction. In our problem, there are two cases of false prediction. Case (1):

Algorithm 1: Cost-Sensitive Random Forest Algorithm

Input : a set of n instances D, a set of attributes A
Output: a random forest F of m decision trees

1 **for** *each i from 1 to m* **do**
2 create a instance set D_i by sampling, with replacement, n instances from D
3 randomly select a subset of attributes $A_i \subset \mathcal{A}$
4 create a decision tree $h_i = \text{TreeGenerate}(D_i, A_i)$
5 calculate cost-sensitive error rate e_i for tree h_i according to (1)
6 **for** *each decision tree h_i* **do**
7 calculate a weight $w_i = \frac{e_i^{-1}}{\sum_j e_j^{-1}}$

Algorithm 2: TreeGenerate(D_i, A_i)

1 create a root node
2 randomly select a subset of attributes $A' \subset A_i$
3 select the attribute a^* with maximum information gain, i.e., $a^* = \arg\max_{a \in A'} gain(D_i, a)$
4 **for** *each possible value α_j^* of a^** **do**
5 create a branch
6 maintain a set D_i^j by including instances whose value of attribute a^* is α_j^*
7 **if** $D_i^j = \emptyset$ **then**
8 create a leaf node on this branch, and set it as the class with most number of instances
9 **else**
10 TreeGenerate$(D_i^j, A' - \{a^*\})$

if an offloadable packet is identified as unoffloadable, it will be immediately transmitted via cellular network interface, potentially leading to higher energy consumption; Case (2) is identifying unoffloadable data as offloadable, which is more serious because it may affect user experience and incur data retransmission by falsely delaying real-time transmissions. To minimize the probability of case (2), we propose a novel cost-sensitive random forest algorithm whose basic idea is as follows.

Given a set of training data, it constructs a number of decision trees, each of which is a tree-like flowchart where leaf nodes denote decisions (i.e., offloadable or unoffloadable), and non-leaf nodes represent tests on attributes (e.g, whether a packet belongs to a TCP flow or not). An example is shown in Fig. 3. In contrast to the traditional random forest algorithm that treats these decision trees identically, we associate a weight to each of them, indicating the sensitivity of error cost. After training, our algorithm can quickly determine whether the arriving data are offloadable or not by combining the decisions of all trees via weighted voting.

The pseudo codes of the training algorithm are shown in Algorithm 1. Its input contains a set of attributes A, and a training data set $D = \{(x_1, c_1), (x_2, c_2), ..., (x_n, c_n)\}$. Each element (x_i, c_i), also called an instance, consists of a vector x_i of attribute values, and a class c_i indicating if the corresponding packet is offloadable or unoffloadable. Given D and A, we create m decision trees in the for loop from line 1 to 5. In each iteration, we sample n instances with replacement from D to create a new instance set D_i. Then, we randomly select k attributes and maintain them in set A_i. We set k according to [21] that suggests the number of attributes used for training a decision tree to be $k \approx \log_2 |A|$. Based on A_i and D_i, we create a decision tree by invoking the function TreeGenerate, as shown in Algorithm 2, which returns a decision tree represented by h_i. After that, a parameter e_i called cost-sensitive error rate is calculated for this decision tree as follows:

$$e_i = \frac{1}{n}\left(\sum_{x_j \in D^o} \psi \mathbf{1}(h_i(x_j) \neq c_j) + \sum_{x_j \in D^u} \phi \mathbf{1}(h_i(x_j) \neq c_j) \right) \tag{1}$$

where D^o and D^u ($D^o \cup D^u = D$) include instances whose class should be "unoffloadable" or "offloadable", respectively. The indicator function $\mathbf{1}(h_i(x_j) \neq c_j)$ outputs 1 if $h_i(x_j) \neq c_j$; or zero otherwise. Note that $h_i(x_j)$ characterizes the output of decision tree with input x_j. The parameters ϕ and $\psi = 1 - \phi$ denote the weights of false prediction case (1) and case (2), respectively. To minimize the probability of falsely identifying unoffloadable packets as offloadable, we should give more weights to case (2) by setting $\phi > \psi$. The optimal values of ϕ and ψ will be given in the numerical study in Section VI. After the construction of m decision trees, we calculate tree weights in for loop in lines 6 and 7.

As shown in Algorithm 2, we create a decision tree by initializing a root node and a subset of attributes $A' \subset A_i$. Then, we select the attribute a^* with the maximum information gain [21] that is calculated by:

$$gain(D_i, a) = Ent(D_i) - \sum_{\alpha_j} \frac{|D_i^j|}{|D_i|} Ent(D_i^j) \tag{2}$$

where D_i^j is a set of instances whose value of attribute a is α_j. Given an instance set D_i, its entropy can be calculated by:

$$Ent(D_i) = -p_u \log p_u - p_o \log p_o \tag{3}$$

where p_u and p_o denote the portions of unoffloadable and offloadable instances, respectively, in set D_i.

After training, given x, our constructed random forest, represented by F, can quickly return a class via weighted voting:

$$F(x) = \arg \max_{c \in \{c_u, c_o\}} \sum_{i=1}^{m} w_i \mathbf{1}(h_i(x) = c) \tag{4}$$

where c_u and c_o denote the class of unoffloadable and offloadable, respectively. The random forest $F(x)$ is periodically retrained using the latest data accumulated in the local storage of controller, so that our system can work with high accuracy as users move to different environment.

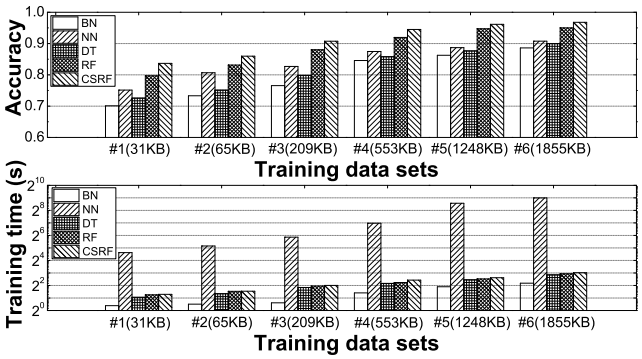

Fig. 4. Performance of machine learning algorithms under different data sets

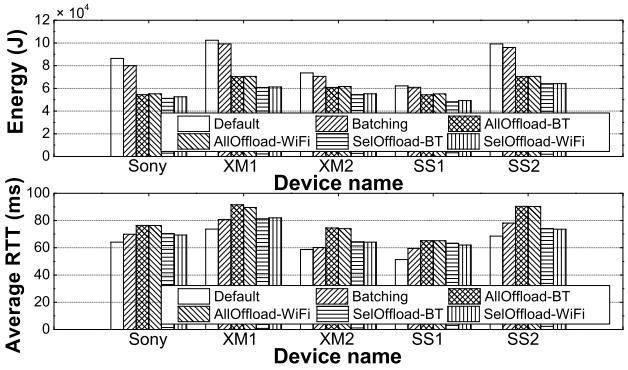

Fig. 5. Performance of offloading approaches on different devices

VI. Performance Evaluation

In this section, we evaluate the performance of our proposals by conducting experiments on a small-scale testbed consisting of 5 Android smartphones, followed by large-scale simulations on a customized network simulator.

A. Experiments on testbed

We build a small-scale testbed by installing the middleware on 5 Android smartphones, including one SONY L36h (labeled as SONY), two XiaoMi Notes (labeled as XM1 and XM2), and two SAMSUNG Galaxy S3s (labeled as SS1 and SS2). They are held by 5 users, respectively, who walk around in the same floor of our research lab. To get training data, we collect traffic traces and label them using the offline method proposed in [7].

We first study the accuracy of our proposed CSRF algorithm by comparing it with four well-known machine learning algorithms: C4.5 decision tree (DT), traditional random forest (RF), Bayesian network (BN), and neural network (NN). We collect 6 traffic traces from different smartphones in the testbed as training data sets. The results of 6 training sets with different sizes (31KB, 65KB, 209KB, 553KB, 1248KB, and 1855KB) are shown in Fig. 4. We observe that accuracy of all algorithms increases as the growth of training data size, and CSRF always outperforms all other algorithms. Even using the smallest data set, CSRF can achieve over 80% accuracy, higher than the fastest BN by 1.6 times.

The training time of these machine learning algorithms using different data sets are also shown in Fig. 4. BN is the fastest one that takes only 1.5 seconds to deal with the largest data set, but it is not preferred due to low accuracy. RF and CSRF have close speed, but much faster than NN. Note that both RF and CSRF take a bit longer training time than DT, but are superior to DT in accuracy because RF algorithms avoid overfitting, which exists in DT, with the bagging idea. To train the smallest data set (i.e., 31KB data generated within one hour), CSRF takes only about 1.1 seconds. Its training time is less than 3 seconds for the largest data set. Since we periodically retrain the packet classifier by every 10 or 30 minutes, CSRF is fast enough to be applied in practice.

We then study the energy consumption and average RTT of smartphones. For comparison, we consider 4 different offloading approaches: (1) the default one without batching and offloading; (2) batching [2]; (3) traditional offloading (denoted by AllOffload) that offloads whenever offloading links emerge [5]; and (4) our proposed selective offloading (denoted by SelOffload). For AllOffload and SelOffload, we show the results of using Bluetooth (BT) and WiFi as the offloading links, respectively. As shown in Fig. 5, SelOffload-BT always consumes the minimum energy in all smartphones, and it can save at most 16% energy of AllOffload in XM1 smartphone. The energy consumption of SelOffload-WiFi is a little bit higher because WiFi is more energy-hungry than Bluetooth. The minimum RTT can be achieved by default approach since it immediately transmits all packets via cellular network interface. Our proposed SelOffload has close performance with the default approach, but much lower RTT than AllOffload, with at most 22% improvement in SS2 smartphone.

B. Large-scale simulations

We consider a number of mobile users that are randomly distributed within a 1km $\times$ 1km square region as the *Poisson point process* (PPP) [22]. Mobile users move according to the random walk model [23] and their average speed is 1.1m/s. We simulate traffic generation of mobile devices according to traffic traces collected from our testbed. We first study the energy consumption of different approaches in 100-node networks. As shown in Figs. 6, our proposed SelOffload significantly outperforms other approaches. There are about 80% mobile devices whose energy consumption is less than 0.6×10^5J, while this portion is only 20% when AllOffload is used in Fig. 6. We also show the average energy consumption and delay of all devices in Fig. 7. In both networks, SelOffload can reduce about 27% energy of AllOffload, and about 45% energy of batching. Since our proposed SelOffload only postpones transmission of offloadable data, it is able to reduce over 60% average delay compared with the AllOffload approach.

VII. Conclusion

In this paper, we propose an on-the-fly selective offloading approach and implement it as a smartphone middleware

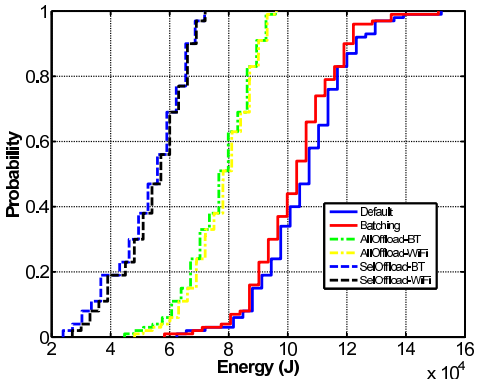

Fig. 6. The CDF of energy consumption in a 100-node random network.

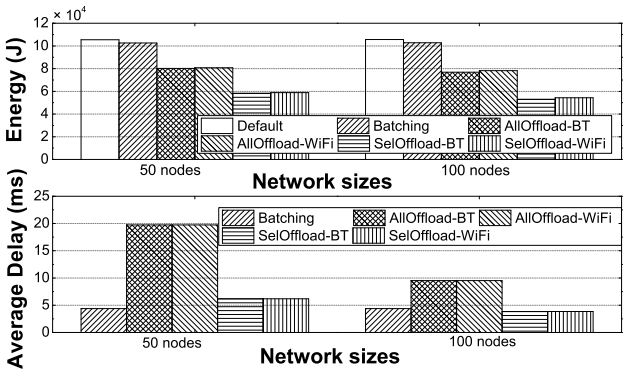

Fig. 7. Average performance of different offloading approaches.

consisting of a packet classifier and a traffic scheduler. The packet classifier can identify offloadable packets using a light-weight machine learning algorithm exploiting unique context information of smartphones. Experiments on both testbed and simulator show that our proposal outperforms state-of-the-art offloading approaches.

ACKNOWLEDGMENT

This research is supported by the Program of International S&T Cooperation under grant No. 2015DFE12860, the National High-tech Research and Development Program of China (863 Program) under grant No.2015AA01A203 and the Science and Technology Planning Project of Guangdong Province in China under grant No.2016B030305002.

REFERENCES

[1] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (MobiCom)*, Istanbul, Turkey, 2012, pp. 317–328.

[2] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC)*, Chicago, Illinois, USA, 2009, pp. 280–293.

[3] D. Zhang, Y. Zhang, Y. Zhou, and H. Liu, "Leveraging the tail time for saving energy in cellular networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 7, pp. 113–122, July 2014.

[4] W. Hu and G. Cao, "Quality-aware traffic offloading in wireless networks," in *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Philadelphia, Pennsylvania, USA, 2014, pp. 277–286.

[5] W. Hu and G. Cao, "Energy optimization through traffic aggregation in wireless networks," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, April 2014, pp. 916–924.

[6] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Pittsburgh, PA, USA, 2013, pp. 29–40.

[7] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck, "Screen-off traffic characterization and optimization in 3G/4G networks," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference (IMC)*, Boston, Massachusetts, USA, 2012, pp. 357–364.

[8] *Periodic transfers. http://developer.att.com/application-resource-optimizer/docs/best-practices/periodic-transfers*.

[9] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys)*, Bern, Switzerland, 2012, pp. 29–42.

[10] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby, "Smartphone energy drain in the wild: Analysis and implications," in *Proceedings of International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Portland, Oregon, USA, 2015, pp. 151–164.

[11] X. Lei, Z. Tang, P. Li, H. Jin, S. Guo, X. Liao, and F. Lu, "Energy minimization for cellular network interfaces with dynamic link quality," in *Proceedings of 24th International Conference on Computer Communication and Networks (ICCCN)*, Aug 2015, pp. 1–7.

[12] T. Zhang, X. Zhang, F. Liu, H. Leng, Q. Yu, and G. Liang, "etrain: Making wasted energy useful by utilizing heartbeats for mobile data transmissions," in *Proceedings of 35th IEEE International Conference on Distributed Computing Systems*, Columbus, OH, USA, 2015, pp. 357–364.

[13] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding, "Cool-tether: Energy efficient on-the-fly wifi hot-spots using mobile phones," in *Proceedings of CoNEXT*, Rome, Italy, 2009, pp. 109–120.

[14] *3GPP TR 37.804, http://www.qtc.jp/3GPP/Specs/37804-b00.pdf, 2006*.

[15] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Low Wood Bay, Lake District, UK, 2012, pp. 225–238.

[16] B. Han, P. Hui, V. Kumar, M. Marathe, J. Shao, and Srinivasan, "Mobile data offloading through opportunistic communications and social participation," *IEEE Transactions on Mobile Computing*, vol. 11, no. 5, pp. 821–834, May 2012.

[17] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, Apr. 2006.

[18] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Banff, Alberta, Canada, 2005, pp. 50–60.

[19] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.

[20] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[21] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[22] S. N. Chiu, D. Stoyan, W. S. Kendall, and J. Mecke, *Stochastic geometry and its applications*. John Wiley & Sons, 2013.

[23] S. Bandyopadhyay, E. J. Coyle, and T. Falck, "Stochastic properties of mobility models in mobile ad hoc networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 11, pp. 1218–1229, Nov. 2007.