A Novel Matching Theory-Based Framework for Computation Offloading in Device-to-Device Communication

Dheeraj Mittal*, Udit Narayana Kar†, Debarshi Kumar Sanyal‡
*†‡School of Computer Engineering, KIIT University, Bhubaneswar, Odisha – 751024, INDIA
Email: *dheeraj.mittal0123@gmail.com, †uditnarayankar@gmail.com,
‡debarshisanyal@gmail.com

***Abstract*—The increasing demand for multimedia mobile applications creates challenges for smartphones that have low memory or slow processor. In this paper, we explore how local cooperation among the devices can help alleviate this problem to some extent. Local cooperation can be achieved through device-to-device communication which is expected to be a significant feature of next generation cellular networks. The cooperation can be achieved by having smartphones with insufficient resources offload their jobs to neighboring smartphones with higher computational capability. We model it as a stable matching problem. We also propose a novel flexible framework and simple distributed algorithms to pair mobile devices of heterogeneous capabilities.**

***Index Terms*—Computation offloading, Device-to-device communication, Cellular network, Matching, Gale-Shapley algorithm, Smartphone computing**

I. Introduction

Recent times have witnessed tremendous growth in mobile subscriber base. This is primarily driven by the emergence of smartphones together with high data rates, mobile Internet and multifarious mobile applications. But this was not accompanied by a similar stellar improvement in battery technology. Thus, mobile devices that run computation-heavy tasks or make too much use of the user interface quickly drain their power. A common feature of the mobile phone user community is the heterogeneity of the devices they use. While some of them possess high-end CPU or large memory, others are seriously resource-constrained. For example, a low-priced Swipe Elite 2 smartphone has 1 GB RAM and a 1.3 GHz quad-core processor [1] while expensive Samsung Galaxy S8 Plus smartphones feature 4 GB RAM and octa-core (2.3 GHz Quad + 1.7 GHz Quad) application processors [2]. Many software applications cannot run comfortably on resource-constrained smartphones. A natural question is, can users in such a diverse community cooperate so that the less capable devices may utilize the potential of more capable ones?

We believe the emerging device-to-device (D2D) communication technology can be a promising paradigm to address this issue in 4G and beyond 4G networks [3]. D2D communication enables direct communication between two mobile devices in proximity without traversing the base station (BS) and the core network. The BS may, however, exercise some control on the management of D2D sessions. D2D communication may use license-free spectrum (like Wi-Fi) or the licensed cellular spectrum (like LTE networks). In the second case, the mobile operator may reserve a part of the spectrum for D2D users, or simply allow the D2D users to exploit unused portions of the spectrum opportunistically. It leads to greater spectrum efficiency, higher throughput and lower end-to-end delay. It has potentially many applications in public safety systems, local data dissemination and range extension of cellular networks. Recently, D2D communication in cellular networks has drawn much attention from both industry and academia. Fig. 1 shows the basic architecture of D2D communication.

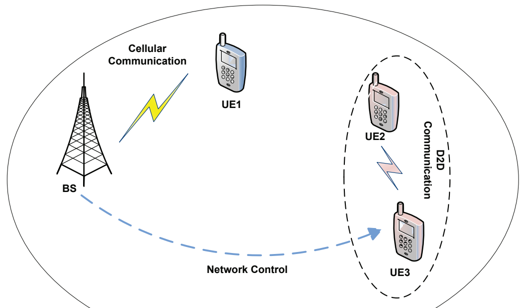


Fig. 1. D2D communication

D2D communication can be used by smartphones to offload part of their tasks to other smartphones in their vicinity (e.g., in an office room or in a bus). This cooperative interaction may prove to be a boon for resource-constrained devices. Thus, the problem becomes one of pairing resource-rich and resource-constrained devices for effective computation offloading. We model it as a stable matching (or stable marriage) problem and adapt the well-known Gale-Shapley algorithm [4] to solve it. This algorithm is one of the cornerstones of matching theory which is concerned with generation of stable allocations of goods among consumers especially when the goods are heterogeneous, indivisible and of limited quantity [5], [6]. We limit ourselves to two-sided matching [7] although the theory encompasses various other possibilities.

Our solution is presented in the form of a generic framework that can be augmented and enriched in different ways. For example, one may like to award incentives to a resource-rich device to act as surrogate. A straightforward means is to stipulate that its service will be charged, i.e., devices offloading tasks to surrogates must pay the latter. The amount a server charges or alternatively, the amount a client is willing to pay may be easily incorporated in our logic to decide the matched pairs. More specifically, participants can collect important characteristics of their potential partners and rank them *a priori*. This ranked order (or preference list) maintained by a device is an input to our algorithm. Thus, the proposed framework is powerful enough to model several scenarios while remaining a simple and intuitive one. We show through measurements (taken with an Android app we developed) that the time consumed by the same computation varies widely across smartphone brands and hence, our scheme can lead to appreciable benefits to users.

II. Related Work

A lot of research has gone into analysis and design of techniques for offloading data and tasks in mobile computing [8]. Mobiles can offload tasks to other mobiles or more powerful servers in the cloud and thus, augment their capabilities with external resources. This helps devices overcome their limitations in terms of computation time, memory consumption and energy. The applications to be offloaded should be adaptively split and suitable parts should be transmitted to remote devices. Such applications are generally called elastic mobile applications. The offloading software must function transparently to give developers the illusion as if they are programming on more powerful mobile devices. Researchers have formulated many guidelines to take offloading decisions that (1) improve performance and (2) save energy. The Cuckoo framework allows users to develop simplified smartphone applications that support dynamic runtime system that can decide at runtime whether a part of the application should be run locally or offloaded to a remote cloud server [9]. The seminal MAUI architecture computes the energy savings possible by fine-grained code offload to a cloud server and accordingly identifies and migrates code dynamically to remote machines [10]. For different sensors in the smartphones different applications are used. Understanding remote execution of applications and writing code that allows efficient offloading are a challenge to application developers. The Android-based DPartner tool automatically analyzes an application bytecode for discovering the parts of an app that are worth offloading, then rewrites the bytecode to support on-demand offloading [11]. The authors in [12] create a software clone of a mobile device in the cloud. They propose two new techniques, off-clone and back-clone. Off-clone performs computational offloading to the clone on the fly and back-clone restores user's data and apps from the clone. They analyze the energy and communication overhead of synchronizing the clones.

The above works mainly focus on offload from mobiles to servers in the cloud. D2D communication in LTE-Advanced opens up a new facet to offloading that can occur exclusively among peer mobile devices. In content delivery networks, users can use D2D links to offload traffic. Xu *et al.* assume that social relationships among users indicate similarity of their preferences for content [13]. They attempt to match content providers with content consumers as well as match D2D links (between the providers and consumers) with spectrum resources. Cao *et al.* advance a framework for joint data and task offloading from one mobile device to another using D2D links [14]. They posit that a device (relay) which has sufficient data usage capacity but is running low on battery should be able to offload the task and data to a nearby device (surrogate) that has sufficient energy budget but low data usage capacity. They propose a matching-based and a game-theory-based solution to form D2D pairs that would participate in the offloading. The first solution is a centralized algorithm. It involves constructing a weighted bipartite graph where the relays and the surrogates form the disjoint vertex sets and the weighted average of the costs of data and task offloading from a relay to a surrogate forms the weight of the connecting edge between them. They apply the Hungarian matching algorithm [15] to find a matching that has the minimum sum of edge weights. The second is a faster distributed algorithm based on coalitional games. Matching algorithms have also been applied in D2D communication for other purposes like mode selection [16] and resource allocation [17], [18]. The reader may refer to [19], [20] for tutorials on matching theory and its applications in wireless networks.

Our approach has goals different from the previous ones. It is closest in motivation to [14] but unlike it, we intend to opportunistically exploit the heterogeneity in quality of smartphones distributed among neighboring users. Toward this end, we propose a stable-matching algorithm. It is a generic framework where desirable preference lists can be easily plugged in.

III. Problem Statement

Current times have seen the explosion of smartphones although it is clearly observable that not all users have high-end varieties. Hence, the computational powers of different users vary to a large extent. So it would be useful for resource-constrained devices (*clients*) to utilize the capabilities of their resource-rich peers (*servers*). We intend to develop a framework where a mobile user can easily offload computation to a suitable smartphone in his proximity using D2D communication. Naturally the number of clients and servers may be unequal. A client may find some servers unacceptable (e.g., because they do not have the desired computational capability), and a server may not like to work for some clients (e.g., because their past history is not commendable). We assume a device acts as either a client or a server but never both at the same time. Regarding the wireless channel, we assume communication is through some form of TDMA; time is slotted and devices are synchronized on slot boundaries. Further, a device cannot receive and transmit at the same time and multiple concurrent transmissions are destroyed completely.

Formally, assume the set of resource-constrained devices is D_1 and the set of resource-rich devices is D_2 and the sets are finite and non-overlapping. Number of elements in D_1 and D_2 may *not* be same. Each device in each of the above sets has a preference list which is defined as a *strict* ordering of a subset of elements of the other set. Thus, preference lists may be *incomplete*. A *matching* M is defined as a set of ordered pairs $\{< d_1, d_2 >: d_1 \in D_1 \wedge d_2 \in D_2\}$ such that each $d_1 \in D_1$ appears in at most one pair in M and each $d_2 \in D_2$ appears in at most one pair in M. A pair $< d_1, d_2 >$ is *acceptable* if each member of the pair appears in the preference list of the other. A *blocking pair* m is defined as a pair $< d_1', d_2' > \notin M$ such that the following 3 conditions hold simultaneously: (1) $< d_1', d_2' >$ is an acceptable pair, (2) either d_1' is unmatched or strictly prefers d_2' to its current partner, and (3) either d_2' is unmatched or strictly prefers d_1' to its current partner. A matching M for which there is at least one blocking pair is *unstable*, otherwise it is *stable*. A *maximum stable matching* is a stable matching with the largest possible number of pairs. It is a natural and desirable solution. Our aim is to find a maximum stable matching given the sets D_1 and D_2 and their preference lists.

IV. Proposed Solution

A. Background on Gale-Shapley Algorithm

1) Basic model and algorithm: The problem of pairing clients and servers can be solved using Gale-Shapley algorithm [4], [21]. This algorithm is typically described in the context of marriage where there are two finite and disjoint sets, one of n men and another of n women. Initially all members of both sets are free and each has a preference list that contains a strict ordering of *all* members of the other gender. Here a *stable matching* M is a collection of pairs $\{< m, w >\}$ where m denotes a man and w denotes a woman such that there is no blocking pair $< m', w' > \notin M$ in which *both* m' and w' prefer each other to the partners they currently have in M. The algorithm proceeds in stages. In each stage, an unengaged man proposes to a woman who ranks highest in his preference list and to whom he has *not* proposed yet. If she is free, she *provisionally* agrees to the proposal and pairs up with him. If she is engaged but prefers him to her current partner, she comes out of her current engagement and pairs up with this man. The process is repeated as long as there is a free man who has not proposed to all women. This algorithm always terminates, generates a stable matching in time $O(n^2)$ and everyone gets married. In an execution of the Gale-Shapley algorithm, the free men can propose in any order but this is inconsequential in the sense that all executions of the algorithm return the same stable matching. When men propose, the stable matching returned is man-optimal, i.e., each man is married to the best (i.e., his highest ranked) partner he can have in any stable matching. It is also woman-pessimal, i.e., each woman is married to the worst (i.e., her lowest ranked) partner she can have in any stable matching. If instead, the women propose, the algorithm would return a woman-optimal (and man-pessimal) stable matching.

2) Unequal numbers of men and women: Consider the variant where the two sets have different cardinalities, say, n men and k women [5], [22]. Not all persons can be married now. Clearly, a blocking pair $< m', w' > \notin M$ now means a pair in which (1) either m' is free or strictly prefers w' to his current partner, and (2) w' is free or strictly prefers m' to her current partner. The algorithm still returns a maximum stable matching in which all members of the smaller set are matched and $|n - k|$ members of the larger set are unmatched. The larger set is partitioned into two subsets: (1) members that have partners in all stable matchings, and (2) members that do have partners in any stable matching.

3) Incomplete preference lists + unequal set sizes: Another important extension of the problem is that of incomplete preference lists where a person finds some persons of the other gender unacceptable [5], [22]. Now a blocking pair $< m', w' > \notin M$ must additionally obey a third property: $< m', w' >$ should be an acceptable pair, i.e., each member of the pair must appear in the preference list of the other. The Gale-Shapley algorithm carries over to this case with the obvious clause that a man does not propose to a woman he finds unacceptable and a woman rejects any proposal from a man she finds unacceptable. As in previous cases, the algorithm returns a maximum stable matching. Men and women are each partitioned into two subsets: (1) those who have partners in all stable matchings, and (2) those who do not have partners in any stable matching.

B. Adaptation of Gale-Shapley Algorithm for Matching Devices

When the Gale-Shapley algorithm is adapted to the current setting, men and women correspond to clients and servers respectively. Our assumption that the number of clients and the number of servers are unequal is a very reasonable one and models reality closely since these numbers depend purely on the local population of mobiles. A client has a technical requirement like high CPU frequency or large memory or a specific software (e.g., to run a video-processing program). It may choose only those servers that possess CPU, memory and software satisfying its requirements. Then it ranks the chosen servers strictly. Each server also orders (possibly, a subset of) the clients strictly, say, on the basis of the bids submitted by the clients or the compute time required or other features of the clients. However, we keep the issues of designing preference lists outside the scope of this paper and simply assume that clients and servers have their own preference lists. The sequence of steps executed by the devices to form pairs are shown in Algorithm 1. In every iteration, a free client proposes to the server highest on its preference list. The server chooses the client, irrespective of whether it is currently paired or not, based on whether the client currently proposing is higher on its preference list. This eventually generates a maximum stable matching.

```
Input: Device sets $D_1$ and $D_2$ (along with their
       preference lists)
Output: A stable matching $M$

$M = \phi$;
Set status of all $c \in D_1$ as free;
Set status of all $s \in D_2$ as free;
while (there is a free client $c \in D_1 \wedge c$ has not proposed
 to all servers in its preference list) do
    $s \leftarrow$ first server ($\in D_2$) on $c$'s preference list to
     whom $c$ has not proposed yet;
    if $c$ is present in preference list of $s$ then
        if $s$ is free then
            $M \leftarrow M \cup < c, s >$ /* $s$ accepts $c$ */;
        else
            /* some pair $< c', s >$ already exists in $M$ */
            if $s$ prefers $c$ to $c'$ then
                $M \leftarrow M \setminus < c', s >$ /* $s$ rejects $c'$ */;
                $M \leftarrow M \cup < c, s >$ /* $s$ accepts $c$ */;
            else
                /* $s$ rejects $c$ */
            end
        end
    else
        /* $s$ rejects $c$ */
    end
end
return $M$;
```

Algorithm 1: Matching algorithm for clients and servers

C. Proposed Framework for Computation Offloading

We propose a computation offloading framework that has three consecutive phases. We assume clients and servers do not enter or exit the system during the phases.

In the *first* phase, essential information for formation of preference lists is disseminated in the network. For example, servers may announce their capabilities and clients their bids, etc. We intend the next phase to run a *distributed* version of Algorithm 1 faithful to our channel model. Although the devices in D_1 are not required to propose in a specific order, they must follow some *sequential* order since our channel model does not allow concurrent transmissions. So a central controller (say, base station) must prepare a fixed transmission schedule π for use by devices in D_1 in phase 2. There are at most $T = |D_1||D_2|$ iterations - each iteration having a proposal from a device $d_1 \in D_1$ to a device $d_2 \in D_2$, an accept/reject message from d_2 to d_1 and possibly, a reject message from d_2 to some other engaged node in D_1. (More precisely, if p_i is the size of the preference list of device $d_i \in D_1$, the number of iterations is no more than $\sum_{i=1}^{|D_1|} p_i$.) We can encode the receiver information in the payload and merge the last two message types into a broadcast message. This means each request time slot is followed by a slot for (broadcast) acknowledgement from D_2. Clearly at most T slot pairs (i.e., pair of request and response slots) are needed. The base station can generate a transmission schedule π by randomly assigning $|D_2|$ specific slots (or exactly p_i slots if p_i is known to it) to each device $d_i \in D_1$. This schedule is broadcast to the devices in D_1 as the last operation of the first phase.

In the *second* phase, clients send requests to servers and servers respond to them by accepting or rejecting the requests. Clients can propose in the request slot of each slot pair using Algorithm 2. In other words, a client checks if it is scheduled to transmit in that slot. If so and it is free and has not proposed to all servers in its preference list, it transmits to its most preferred server to which it has not proposed yet. In a response slot, a server responds using the logic in Algorithm 3. The procedure `sendMsg` takes a `sender_name` followed by `destination_count` followed by `<destination_name, PROPOSE|ACCEPT|REJECT>` tuples occurring `destination_count` times. If `destination_count` is 1, a unicast message is sent to the destination. Otherwise a broadcast message is sent with the payload indicating the different destinations and the corresponding message bodies. Since a server can revise its decision by coming out of its current engagement and pairing with another device, a client can know its surrogate only at the end of this phase.

```
Input: Transmission schedule $\pi$ of a client
Output: The client either sends PROPOSE message to a
        potential server or no message at all

$c \leftarrow$ this device;
if $c$ is scheduled to transmit in this slot then
    if $c$ is free and $c$ has not proposed to all servers in
     its preference list then
        $s \leftarrow$ first server ($\in D_2$) on $c$'s preference list to
         whom $c$ has not proposed yet;
        sendMsg($c$, 1, $s$, PROPOSE);
    else
        /* $c$ is engaged or has exhausted its preference
         list */
    end
else
    /* $c$ is not scheduled to transmit in this slot */
end
```

Algorithm 2: Algorithm used by a client in each request slot

In the *third* and final phase, each matched client sends its computational task and data to its corresponding server. The latter executes it and returns the results.

It is clear that the last two phases comprise completely distributed algorithms executed by the devices. A sequence of message flows corresponding to the proposed algorithm is shown in Fig. 2.

V. Implementation and Measurements

We developed an Android app using Google Firebase [23] platform to offload computation from a smartphone to another nearby smartphone. Our goal is to find out whether computation times vary appreciably across different grades of mobiles.

```
Input: Message queue $Q$ of a server
Output: The server either sends message to one or more
        clients or no message at all
$s \leftarrow$ this device;
if $Q$ not empty then
    $m \leftarrow$ message from client $c$ in $Q$;
    if $c$ is present in preference list of $s$ then
        if $s$ is free then
            sendMsg($s$, 1, $c$, ACCEPT);
        else
            /* $s$ is currently paired with $c'$ */
            if $s$ prefers $c$ to $c'$ then
                /* Accept $c$ and reject current partner $c'$ */
                sendMsg($s$, 2, $c$, ACCEPT, $c'$, REJECT);
            else
                sendMsg($s$, 1, $c$, REJECT);
            end
        end
    else
        sendMsg($s$, 1, $c$, REJECT);
    end
else
    /* Nothing to do */
end
```

Algorithm 3: Algorithm used by a server in each response slot

The app uses Wi-Fi to connect since D2D communication support is not available in our systems. A snapshot of the system interface is shown in Fig. 3. The leftmost screen shows how a user profile is set up. It is used to inform the app whether the phone will serve as a server or a client. If it is a client, it has to specify a bid value too. We assume each client offers the same bid to all servers. The `Proceed` button is used to complete the registration. The details are uploaded to Firebase Realtime Database. The `Search` button is used to search for a specific device. As the next snapshot in Fig. 3 shows, the list of available servers displayed to a client contains the details of each server in terms of its identifier, its total memory and its available memory. Clients interact with servers to form pairs. The screenshots for this interaction are not shown. The rightmost snapshot shows the interface for a client to send data to the server to which it paired up. The same interface is also used by a client to receive results from the server. Here, we assume the program / task to be executed is already available with the server.

We experimented with Android smartphones from three different vendors. The device specifications are shown in Table I. Here, device P1 is the most high-end device while device P3 is the most low-end device as per the device specifications. We implemented two programs: (1) a primality checker that takes an input integer and tests if it is prime, and (2) a palindrome checker that takes a bit-string and checks if the string reads the same backwards and forwards. The results of running the same program on different mobiles (chosen as servers) are shown

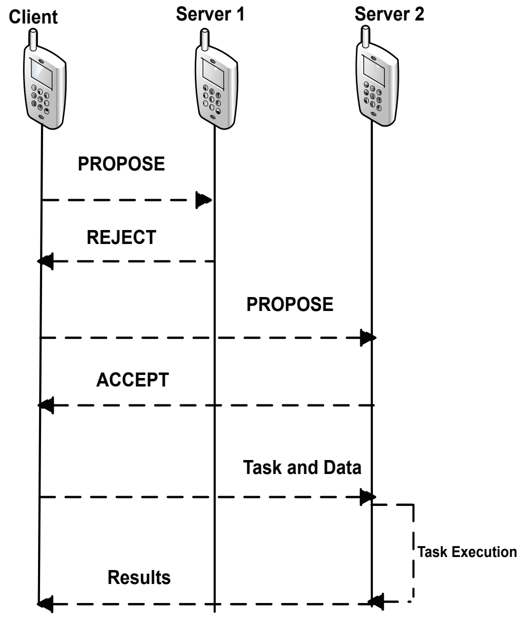
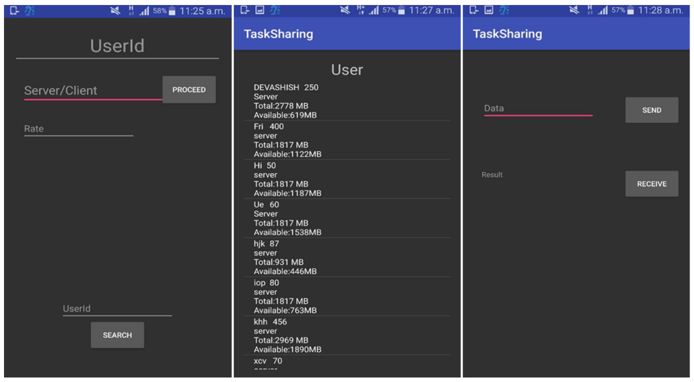


Fig. 2. Message flow diagram showing interactions among devices for computation offloading



User

DEVASHISH 250
Server
Total:2778 MB
Available:619MB

Fri 400
server
Total:1817 MB
Available:1122MB

Hi 50
server
Total:1817 MB
Available:1187MB

Ue 60
Server
Total:1817 MB
Available:1538MB

hjk 87
server
Total:931 MB
Available:446MB

iop 80
server
Total:1817 MB
Available:763MB

khh 456
server
Total:2969 MB
Available:1890MB

xcv 70

Fig. 3. Mobile app to offload computation: (From left) Initial interface to set up user profile, List of servers shown to a client, Interface to send data to and receive result from server

in Tables II, III. It can be observed that the time taken by the LG Nexus 5 mobile is less than that taken by the Karbonn A6 mobile which is less than the time consumed by Idea ID-920 mobile. This shows that computation offloading can result in faster processing.

TABLE I
DEVICE SPECIFICATIONS

Device ID	**Brand**	**Processor**	**RAM**
P1	LG Nexus 5	2.3 GHz (quad-core)	2 GB
P2	Karbonn A6	1 GHz (single-core)	512 MB
P3	Idea ID-920	1 GHz (single-core)	256 MB

TABLE II
TIME IN *milliseconds* TO RUN PRIMALITY CHECKER ON DIFFERENT SMARTPHONES. EXPERIMENT WITH EACH INPUT INTEGER IS DONE THRICE. THE AVERAGE VALUES ARE ALSO SHOWN.

Input integer	Time on P1	Time on P2	Time on P3
5	1.7286	1.8632	2.3081
5	1.7702	2.1558	1.9752
5	1.8425	1.9383	2.2232
(Avg. time ⇒)	1.7804	1.9858	2.1688
5555	2.2941	2.0157	2.3350
5555	2.1703	2.3587	2.4415
5555	2.2109	2.4782	2.5625
(Avg. time ⇒)	2.2251	2.2842	2.44633
555555555	303.6833	316.8265	326.7898
555555555	316.2618	321.5064	336.7539
555555555	325.4091	309.5345	340.3068
(Avg. time ⇒)	315.1181	315.9558	334.6168

TABLE III
TIME IN *milliseconds* TO RUN PALINDROME CHECKER ON DIFFERENT SMARTPHONES. EXPERIMENT WITH EACH STRING IS DONE THRICE. THE AVERAGE VALUES ARE ALSO SHOWN.

Input string	Time on P1	Time on P2	Time on P3
1010	1.8861	1.9060	2.3896
1010	1.9296	2.3833	2.2263
1010	2.0959	2.6472	2.3325
(Avg. time ⇒)	1.9706	2.3122	2.3162
101011	1.5483	2.305	2.1240
101011	1.6539	1.9345	1.9333
101011	1.8112	2.2363	2.4468
(Avg. time ⇒)	1.6712	2.1586	2.1681

VI. Conclusion and Future Work

We proposed a framework for pairing mobile devices to enable computation offloading in a scenario where a mix of high-end and low-end devices are present. Clients and servers maintain preference lists and choose their partners using a match-making algorithm. Our simulations demonstrate that computation times can vary widely across smartphone brands. Therefore, cooperative task execution can be really useful to low-end mobiles.

We have identified several directions for future research. In particular, we aim to consider methods to generate preference lists, consider preference lists that are partial orders over mobiles of the other set, consider the possibility of a server accepting multiple clients or a client offloading to multiple servers, investigate link volatility at a smaller time scale, design algorithms to place bids intelligently, and measure the performance of the algorithm in a large population of heterogeneous smartphones. We believe a full-fledged mobile app based on our framework and incorporating the above aspects can be of great practical utility.

References

[1] Swipe, "Smartphone specifications," http://justswipe.com/the-product/smartphones/, (Accessed on 16/7/2017).

[2] Samsung, "Galaxy s8 / s8+ specifications," http://www.samsung.com/global/galaxy/galaxy-s8/specs/, (Accessed on 16/7/2017).

[3] L. Wang and H. Tang, *Device-to-device communications in cellular networks*, ser. SpringerBriefs in Electrical and Computer Engineering. Springer, 2016.

[4] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[5] M. Niederle, A. E. Roth, and T. Sönmez, "Matching and market design," in *The New Palgrave Dictionary of Economics*, S. N. Durlauf and L. E. Blume, Eds. Basingstoke: Palgrave Macmillan, 2008.

[6] A. Abdulkadiroglu and T. Sönmez, "Matching markets: theory and practice," *Advances in Economics and Econometrics*, vol. 1, pp. 3–47, 2013.

[7] D. Gale, "The two-sided matching problem: origin, development and current issues," *International Game Theory Review*, vol. 3, no. 2 & 3, pp. 237–252, 2001.

[8] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[9] R. Kemp, N. Palmer, T. Kielmann, and H. E. Bal, "Cuckoo: a computation offloading framework for smartphones," in *Proceedings of the 2nd International ICST Conference on Mobile Computing, Applications, and Services (MobiCASE)*. Springer, 2010, pp. 59–79.

[10] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, 2010, pp. 49–62.

[11] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring android java code for on-demand computation offloading," in *ACM SIGPLAN Notices*, vol. 47, no. 10. ACM, 2012, pp. 233–248.

[12] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *Proceedings of the 32nd Annual IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2013, pp. 1285–1293.

[13] C. Xu, C. Gao, Z. Zhou, Z. Chang, and Y. Jia, "Social network-based content delivery in device-to-device underlay cellular networks using matching theory," *IEEE Access*, vol. 5, pp. 924–937, 2017.

[14] Y. Cao, C. Long, T. Jiang, and S. Mao, "Share communication and computation resources on mobile devices: a social awareness perspective," *IEEE Wireless Communications*, vol. 23, no. 4, pp. 52–59, 2016.

[15] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics (NRL)*, vol. 2, no. 1-2, pp. 83–97, 1955.

[16] Y. Cao, T. Jiang, and C. Wang, "Cooperative device-to-device communications in cellular networks," *IEEE Wireless Communications*, vol. 22, no. 3, pp. 124–129, 2015.

[17] O. Semiari, W. Saad, S. Valentin, M. Bennis, and H. V. Poor, "Context-aware small cell networks: how social metrics improve wireless resource allocation," *IEEE Transactions on Wireless Communications*, vol. 14, no. 11, pp. 5927–5940, 2015.

[18] Z. Zhou, K. Ota, M. Dong, and C. Xu, "Energy-efficient matching for resource allocation in d2d enabled cellular networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 6, pp. 5256–5268, 2017.

[19] Y. Gu, W. Saad, M. Bennis, M. Debbah, and Z. Han, "Matching theory for future wireless networks: fundamentals and applications," *IEEE Communications Magazine*, vol. 53, no. 5, pp. 52–59, 2015.

[20] Z. Han, Y. Gu, and W. Saad, *Matching theory for wireless networks*. Springer, 2017.

[21] D. Gusfield and R. W. Irving, *The stable marriage problem: structure and algorithms*. MIT press, 1989.

[22] K. Wayne. Lectures on cos 423: theory of algorithms. http://www.cs.princeton.edu/∼wayne/cs423/lectures/intro-marriage-4up.pdf. (Accessed on 16/7/2017).

[23] Google, "Firebase," https://firebase.google.com/, (Accessed on 16/7/2017).