DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Towards a Collaborative Context-Aware Offloading Scheme in Mobile Cloud Computing

Candidate

Mohammad Abu Snober (1629221)

Supervisor

Prof. Roberto Beraldi

Co-Supervisor

Prof. Andrea Vitaletti

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering in Computer Science

XXIX Cycle

July 2017

Abstract

The use of modern mobile devices over traditional desktop and tab devices is dramatically increased. Mobile devices have limited resources regarding computation power, storage and battery. Moreover, mobile apps market is a huge market with billions of applications. Some of those applications like gaming and pattern recognition are attracted users to be installed. This kind of apps are known as resource-hungry apps due to the need for a powerful device to normally run such apps. From here, researchers work to present efficient solutions to augment the resource-poor mobile devices to be more powerful. One of those solutions is to collaborate among neighboring devices to offload some part(s) of an application. Mobile device cloud (MDC) comes with this idea in order to save time and energy. Moreover, new studies aim to investigate for social factors among mobile devices. Devices sharing some kind of friendship or some common interests are more likely to meet and exchange information.

We study the social factors in a comprehensive way in order to see which factor or combination of factors are the best in conserving execution time and energy. In our investigation study we exploit for some of well-known tracefiles (like *Sigcomm09* and *Unical14*). For those tracefiles we do sanity check and clean the datasets form noise. Then, we define a set of connectivity metrics for a pair of nodes, including: number of contacts, duration of a contact and intercontact time. After that, we quantify for the datasets observations to know which strategy is the best in terms of the number of contacts and total gain acquired by applying the connectivity metrics. The output

of this study indicates that a pair of nodes with at least 4 common interests have the highest number of contacts with about 63% among all contacts. Moreover, adding direct friendship to the pair with at least 4 common interest gives the highest gain regarding the connectivity metrics. with about 64%. Thus, we think that exploiting for the two outputs from this study will give best results in the case of task offloading in a way to save time and energy.

Furthermore, we test for our investigation study numerically against some of proposed offloading algorithm in order to get better understanding of the effects of social factors. Here we use some profiles based on real devices and consider some task capabilities based on real test-beds. Moreover, we design a set of offloading algorithms one of them is based on our investigation study (*S-based*) and the other algorithms exploit for one social factors. Indeed, we test for the replication factor on the offloading performance. Furthermore, we test for medium and high computation tasks against local execution and against our proposed algorithms to see which strategy is the best in terms of number of contact and the gain regarding the connectivity metrics. This help us in answering 2 questions: (i) when to offload a task?, and (ii) what kind of tasks is good to be offloaded?. Numerically, the algorithm based on our investigation study works fine and gives good results regarding time and success rate. The same algorithm saves more than 65% of time regarding local execution and more than 40% regarding *Random* offloading. Moreover, *S-based* gives results close to the lower-bound of the offloading (i.e. *Flooding*). Respect to replication factor, only 2 replicas are enough to reach the maximum performance and to overcome for tasks loss or failure problems.

Finally, we run the whole set of algorithms in real simulation environment using the ONE simulator. Here, we test for number of hosts equivalent to Sigcomm09 (i.e. 76 hosts). In the ONE simulator, we propose that around one-third of the nodes are offloaders each with u tasks and the other hosts are offloadees. Moreover, we propose two cases: the first one considering all devices of the same profile (homogeneous

case) while the other consider the offloadee profiles are higher (heterogeneous case). Simulation results under the ONE simulator are very close to the results obtained by numerical simulation.

Acknowledgment

I would like to thank my supervisor Prof. Roberto Beraldi for his encouragement and thoughtful support during my PhD work. I am grateful to becoming behind me during my study and reserch project here in La Sapienza University. In addition, I would like to express my special appreciation and thanks to my co-supervisor Prof. Andrea Vitaletti for his invaluable advice.

This work is dedicated to my my parents, due to their support for my life and my study from the childhood until now. Without them, my life is nothing. Then, I want to express my thanks for all of my family members; my brothers and sisters for encouraging me to complete my higher education.

This research was made possible through a fellowship by Avempace II project under the umbrella of Erasmus Mundus projects. I would like to thank Prof. Abdallah Al-Zoubi and Prof. Omar Hasan from my home university - Princess Sumaya University for Technology, Jordan. Furthermore, I am grateful to Prof. Graziella Gaglione form the host university - La Sapienza University, Italy.

Contents

List of Tables

List of Figures

Nomenclature

DTN Delay Tolerant Networks

IaaS Infrastructure as a Service

MCC Mobile Cloud Computing

MDC Mobile Device Cloud

MFLOPS mega floating-point operations per second

PaaS Platform as a Service

RTT round trip time

SaaS Software as a Service

VM Virtual Machine

WAN Wide Area Network

Chapter 1

Introduction

Nowadays, most of people have modern mobile devices running Android, iOS or Windows Phone operating systems. Mobile devices are becoming more powerful with higher computation power, higher memory and bigger storage. Moreover, mobile devices are quipped with multiple supporting units, like: wide screens, several sensors, high-resolution cameras, etc. Furthermore, app providers (i.e. Apple App Store and Android Play Store) are overstuffed with different kind of applications. Some type of those applications like gaming, image or pattern recognition apps are attracted by many people to be installed. Actually, such kind of applications require more intensive computation and more power to be executed. [1, 2].

From here the need is raised to extend (augment) the computation power of mobile devices by offloading (migrating of) part(s) of applications to be executed in resource-rich environments. The main goal of mobile offloading is to save time and energy. Due to the increasing request to find efficient ways, in terms of time and energy, mobile cloud computing (MCC) comes with emerged solutions to perform computationally intensive tasks that exceed the limited-resources of mobile devices. The communication between mobile device and cloud provider is typically done in the form of networking services offered by the providers. MCC has a rich environment ideally able to execute intensive-computation tasks in a reasonable time[3, 4].

In fact, there are several scenarios for offloading tasks to the cloud. The first scenario is to send task to a distant cloud. Most popular examples are Amazon EC2 and Windows Azure. Those architectures have massive storage and computation power but suffer from huge delay (i.e. high round trip time "RTT") regarding communication between mobile device and the cloud. Moreover, this type of cloud is susceptible to network intermittent and considered as a very expensive solution [3, 1]. CloneCloud[5], is another solution employs for cloud. CloneCloud is considered as a small powerful cloud able to receive and execute a clone of an application transferred by a mobile device. Another solution more cost effective is cloudlet[6], a resource-rich server placed within a building. Cloudlet solution provide mobile users with real-time and low-latency response. But, what if none of those infrastructures are available at a specific moment. Is there any cost effective solution available? The answer comes with mobile device cloud [7].

MDC can be defined as an ad-hoc cloud constructed among group of mobile devices in order to collaborate in executing some heavy tasks. In MDC the task is initiated in a device called offloader and try to be offloaded on other device(s) called offloadee(s). Actually, the problem here is to find more effective offloading mechanism among mobile devices. Some researches in this field investigate for social factors among mobile devices and try to see the effect regarding performance. This helps in understanding for useful encounters for message forwarding in MDC environment, also it can be useful for better designing of efficient offloading algorithms[8].

1.1 Mobile Cloud Computing

Mobile Cloud Computing (MCC) can be defined as an model that allows mobile devices to expand their limited computing, storage and network resources in order to be run as network services. The process of running those services will be performed on the cloud rather than executing them locally. The main goal of MCC is to allow a mobile device

to offload part(s) of resource-intensive tasks from that device to be quickly executed on the cloud. Furthermore, MCC aim at save energy and time by running offloaded tasks in a rich and comprehensive environment rather than executing them in poor-environment of mobile devices [1, 2]. Furthermore, the computation offloading can be divided into two categories[2, 3]; the first one is about offloading of existing applications (i.e. offloading of the whole application as a single unit then splitting it into several tasks). MAUI, CloneCloud, COMET and ThinkAir are examples follow this strategy. Additionally, there is another kind of offloading models suppose that the process of offloading should be predefined in the development phase of the applications. mCloud and weblet propose this criteria.

1.1.1 MCC components

To know more about mobile cloud computing, we need to define two basic terms: cloud computing and mobile computing. When we combine both terms together we get the term MCC. MCC can be seen as traditional cloud computing (CC) with mobility behavior.

- Cloud computing is a model concerns of employing hardware and software technologies to be in the service of users over a network (typically the Internet). The main goal of this model is: (i) provides for the accessibility anytime everywhere, (ii) gives flexibility and scalability upon user's demand and (iii) be a cost effective solution for computing resources[2, 9]. Furthermore, there are three major layers of services provided by cloud computing as shown in Figure 1.1:

  - Infrastructure as a Service (IaaS), delivers the whole infrastructure (including servers and operating systems) for the demand of users. Amazon Elastic Compute Cloud (Amazon EC2) is a well-known example for this type of cloud service

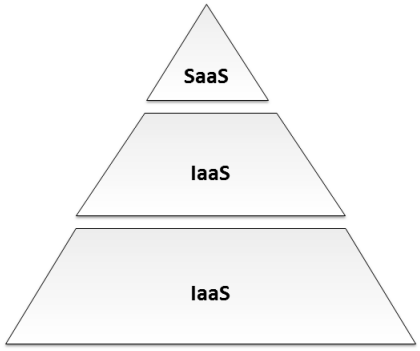

Figure 1.1: Cloud computing layers stack

  - Platform as a Service (PaaS), presents a platform that allows users to create and deploy applications on the cloud without the need of deploying them locally (e.g. Google's App Engine).
  - Software as a Service (SaaS), provides users with the access to a pool of software applications running on the cloud. Google Apps considered as familiar examples for this kind of cloud services.

- Mobile computing, is special kind of computing where the computation is done by some systems. Those systems are physically not fixed in one location (i.e. moved from one place to another). Examples of those systems are: mobile devices, laptop computers, PDAs. Those devices communicating using short-range of communication media like Bluetooth or WiFi forming an ad-hoc network[3, 1, 10]. Those systems are characterized by their resource limitations mainly due to:

  1. Battery size, since the devices within those systems are lightweight and therefore the lifetime is short. Moreover, high-power processing leads to significant

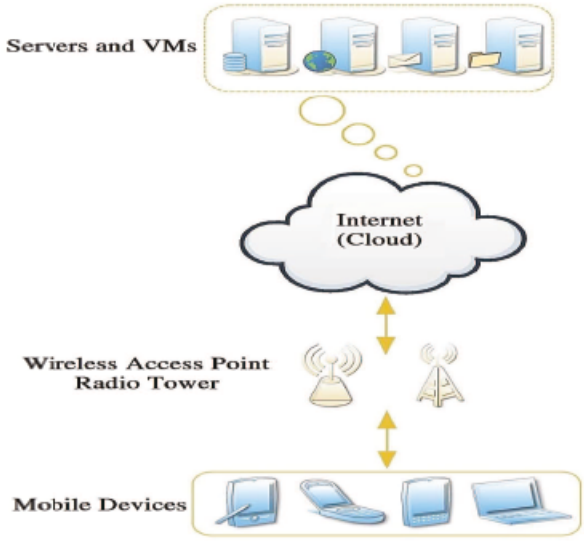

Figure 1.2: MCC architecture [10]

heat which may cause device failure,

2. Limited storage due to their limitation in size, and
3. Poor computational capabilities due to the size and energy limitations of the device.

For that, mobile cloud computing can be considered as a special case of cloud computing where the communication between mobile devices and the cloud is done through wireless connection as shown in Figure 1.2.

1.1.2 Why MCC?

MCC propose a model of how mobile devices using cloud services. Actually, MCC addressing the following challenges[10, 2]:

- Dealing with the limitations of mobile devices, as mentioned earlier mobile devices are considered as poor-resource devices due to battery life and limited computation power. MCC comes with virtualization technique to fix this issue.

- Providing some techniques for partitioning of resource-intensive applications; regarding this, MCC follows two basic strategy as describe in Section 1.1, offloading of the whole application or follow some indications in the application's development phase.

- Pointing for the quality of communications; this is due the intermittent connection among devices in MCC. A suggestion to cope with this issue is to increase the communication bandwidth in a way to reduce delivery time

1.1.3 MCC offloading scenarios

Offloading in MCC can follow some models or scenarios. This is actually depends on the way that a mobile device requests for the cloud services and how this request is handled in the cloud. In other words, MCC models describe the way that mobile devices connect to the cloud and interacts with the cloud services.

The following are the most common MCC models[1, 2]:

- Centralized Cloud; mobile devices can offload parts of their workload to a traditional resource-centric cloud in the form of services. as shown in Figure 1.3. This models suffers from long latency and network delays. It is considered not adequate for real-time mobile applications. Furthermore, it is considered as an expensive solution due to relatively high cost of executing tasks.

- CloneCloud [5]; the idea here is to clone the entire data and applications from the smartphone to nearby computers or data centers. In this architecture, some operations selectively execute on those clones and the results will be reintegrated back to the device.

- Cloudlet[6]; a trusted online resource-rich computer/server or cluster of computers available to nearby devices. Cloudlet provides mobile users with real-time

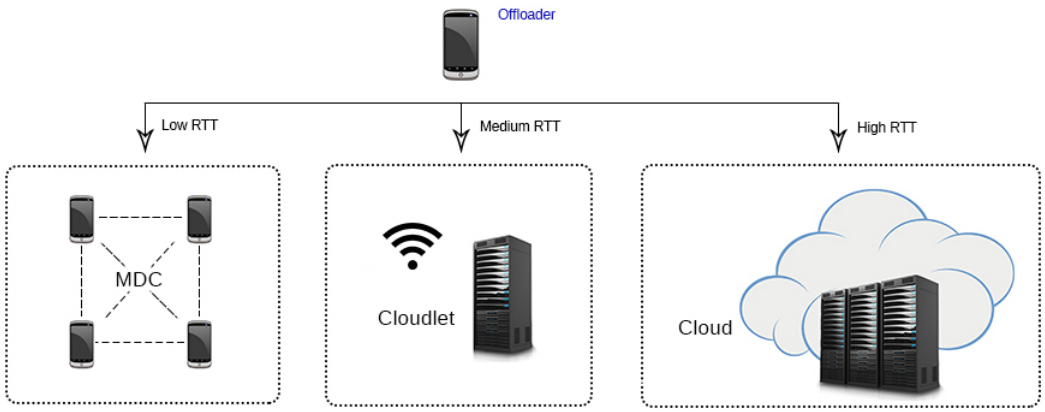

Figure 1.3: MCC scenarios[8]

interactive response. This solution characterized by: low-latency, one-hop and high-bandwidth wireless access.

- MDC [7]; an ad-hoc Mobile-to-Mobile (M2M) cloud. Here, end-to-end task offloading is done between a task initiator (offloder) and an offlodee (task executor). In MDC, neighboring devices are pooled together for resource sharing. In M2M cloud, a task can be processed either by distributing it among shared devices or handled by particular device acting as a server.

1.1.4 Delay Tolerant/ Disruption Networks (DTN)

As we mentioned earlier, ad-hoc mobile devices can be found in a specific region. Those devices do not have any infrastructure to ensure end-to-end connectivity among them. Delay Tolerant/Disruption Network (DTN) architecture is used to create such infrastructure. In addition, DTN architecture implies two facts regarding mobile devices: first, mobile nodes are message carriers used to ensure the delivery of the handled messages. Second, a message can be replicated and distributed to many nodes in order to increase the probability of delivering that message. Moreover, DTN networks are characterized by[11, 12] :

- Intermittent Connectivity
- Long or Variable Delay,
- Asymmetric Data Rates
- High Error Rates

To overcome above issues, a well-known technique called store-carry-forward has been adopted. This technique assumes that each node uses a large buffer that enables it to store messages until it can be forwarded to their destinations.

1.2 Computation offloading

Computation offloading concerns of the process of migrating for an application or part(s) of the application to be executed on the cloud rather than executing them locally. The main goal of computation offloading is to save the application's execution time and consumed energy within acceptable delays[9]. To achieve this goal many solutions for the computation offloading are proposed, here we list the most important of them:

1.2.1 MAUI

A project implemented at Microsoft Research Labs, Los Angeles, called Mobile Assistance Using Infrastructure (MAUI) which introduces a new way for fine-grained mobile offloading. MAUI able to execute computation-intensive applications that exceed the limitations of a mobile device. The main consideration of this project is the energy consumed by mobile devices in a way to overcome the short battery lifetime of such devices. MAUI analyses existing solution for remote task execution and comes with an efficient solution regarding that. MAUI relies on the programmer effeort to determine which task need to be executed locally and whose need to be remotly executed. In the case of a task marked as "remote", the whole code (i.e. program function or method)

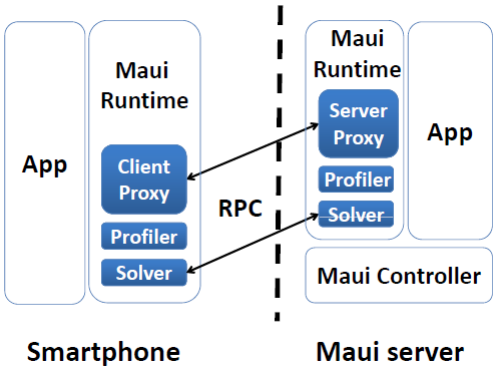

Figure 1.4: MAUI architecture[13]

sent to the infrastructure to be remotely executed there[13]. Figure 1.4 illustrates for MAUI architecture.

MAUI profiler A module used to monitor for different mobile device conditions: energy level and conditions of the wireless in the mobile device, also the requirements for each single method (program unit) regarding execution time and the amount of resources. To predict for energy consumption of each method, Device Profiling comes with a simpler linear model based on number of CPUs require to execute such method. Program Profiling is employed for prediction of program unit (program method) characteristics based on the analysis process of previous executions of the same method. For measuring of wireless network conditions, Network Profiling provide an estimation for the bandwidth, round-trip time (RTT), and packet loss. This is done by sending a TCP packet of specific size to the server and measuring for the transfer duration[13, 1].

MAUI Solver Solver module uses, as input, the data received from Profiler in order to minimizes the total energy consumption of an application. In this module the, methods of an application are divided into two groups: remotly-executed and locally-executed methods. Solver provide solution based on some constraints like programmer's choices and total latency. Furthermore, during the execution precess of an application, Solver module takes the changes in the network conditions to feed the optimization

problem and try to resolves it periodically[13, 3].

Proxy modules MAUI has two basic proxy modules: Client Proxy and Server Proxy, the goal of those modules is to control the data transfer process of the offloaded methods. When the Solver module decides for a migration of a method, the Client Proxy performs serialization process then performs for deserialization process after the call has been performed. After the method has being remotely executed, the Server Proxy performs the serialization state and returns the control back to the mobile device[13].

MAUI Controller Controller module acts as coordinator of the whole system, this module runs on the server side and its goal is to manage for the authentication the resource allocation processes for the incoming requests initiated by the MAUI clients[13].

1.2.2 CloneCloud

Another proposal for MCC introduced by B. Chun et. al. as they propose for CloneCloud. CloneCloud based on application's transition (as threads), here a thread can be executed in a distributed environment. The main goal for partitioning in CloneCloud is to minimize the total execution time and energy consumption through the use of virtualization. This concept concept is illustrated in Figure 1.5. Virtualization aim at creating a remote environment similar to mobile device's environment. This environment able to execute a mobile application remotely. The main concept here is that a virtual machine (VM) has more energy and computation resources that allow programs to be executed quickly[5, 1].

The partitioning process is one of the core processes in CloneClone. This process is not done in realtime completely. This is due to existence of pre-computed partitions in the network. The main function of partitioning mechanism is to create these kind of pre-computed partitions as binary files. Partitioning mechanism indicates for some points placed at the beginning of the thread to be migrated. Those points are called

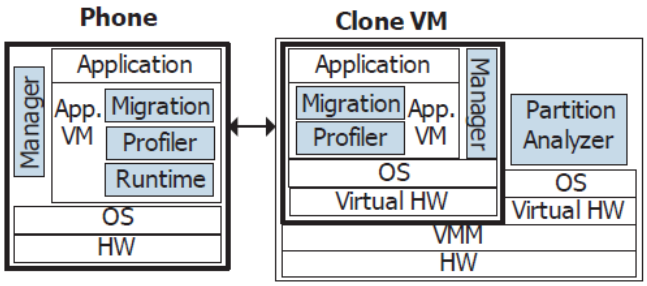

Figure 1.5: CloneCloud architecture[5]

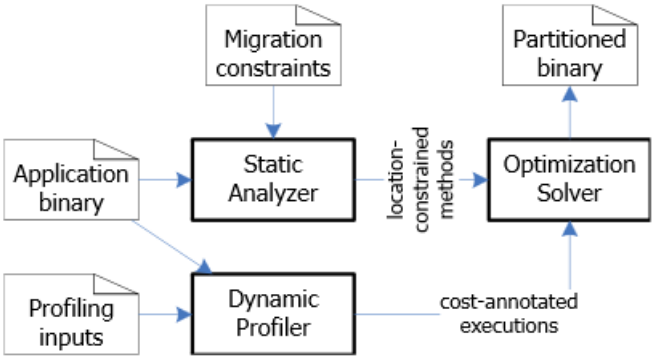

Figure 1.6: CloneCloud partitioning model [5]

migration points. Similarly, re-integration points are placed at the end of those threads [5]. The partitioning process consists of three modules: (i) static analyzer, (ii) dynamic profiler and (iii) optimization solver as shown in Figure 1.6.

Static Analyzer This module performs analysis of the application's code. Moreover, it can change the application by changing the positions of migration and re-integration point. Then, it can provide such information to the Optimization Solver module.

Dynamic profiler The function of this module is to give an indication of the execution cost for the application. This is done by multiple executions of the same application in order to measure for execution time and energy consumption.

Optimization Solver This module proposes some techniques to minimize the costs (i.e. execution time and energy consumption) of some partitions.

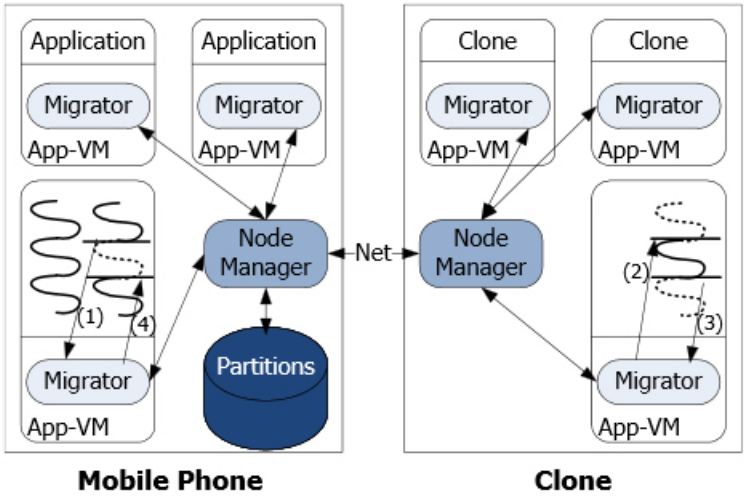

Figure 1.7: CloneCloud Thread management[5]

Thread management CloneCloud relies on multi-threading for executing multiple task concurrently. Thread management module decides weather the thread continue to be executed locally or it is possible to suspend and migrated. In fact, this is done based on the thread requirements. For example, if the thread requires only the use of some mobile device resources (e.g. GPS or UI), it will be executed locally. In the case suspension, Thread Management module provide that thread with a migration point and transfer it to be executed on the cloud. Figure 1.7 illustrates for that module.

1.2.3 Cloudlet

Cloudlet comes as a solution for long WAN latency of the communication process between mobile users and distant clouds. This work is presented by M. Satyanarayanan with the goal of bringing cloud resources to be closer to mobile users in a way to increase the communication bandwidth and reduce the delay. In other words, we can consider cloudlets as small or micro clouds provide cloud services to mobile users within a specific place or building. In fact, Cloudlet employs for an approach called dynamic VM synthesis as shown in Figure 1.8. In this approach the VM consist of two types: VM base and VM overlay. VM base contains information about the application and

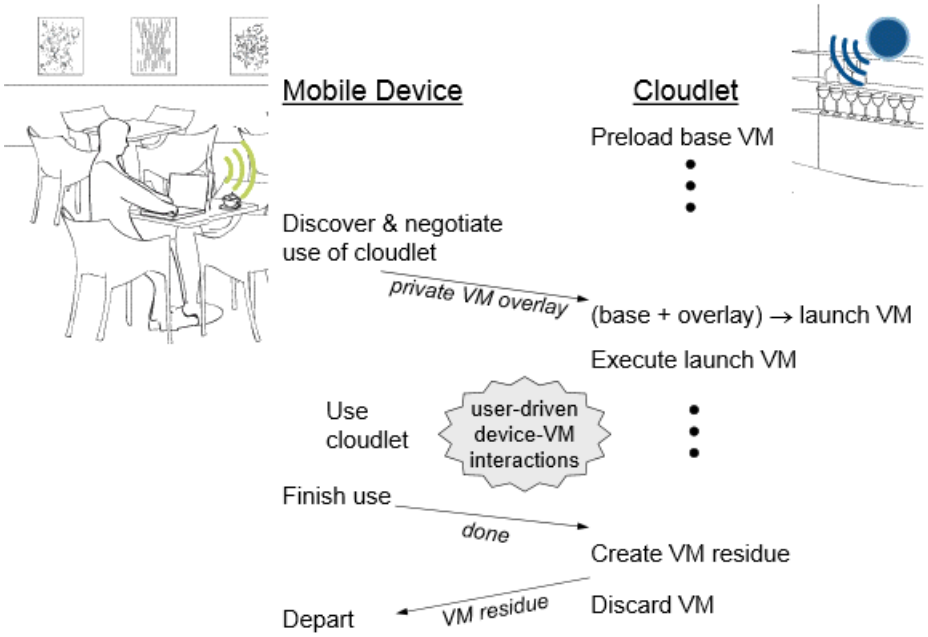

Figure 1.8: Cloudlet architecture[6]

it can be exist in advance in the cloudlet infrastructure. While VM overlay deals with computation-intensive code. In real scenario, the mobile device has just to send the application's VM overlay to the Cloudlet infrastructure. Then the cloudlet match for the corresponding application's VM base in order to execute that application remotly. This scenario implies lower latency and execution cost[6, 9, 3].

Table 1.1 provides a comparison between offloading scenarios.

1.3 MDC collaborative scenarios

MDC present for another offloading scenario, where the cloud not rely on any an external infrastructure. Instead the cloud is formed by nearby devices/stationaries sharing their storage and/or computation reseources. This approach suppose that tasks will be executed quickly with low cost. Moreover, the main advantage for MDC is low latency due to lower RTT between communicating devices. A lot of researcher try to present an efficient offloading way to MDC's devices by investigation of some common behavior

Table 1.1: Comparison between different offloading scenarios

criteria	MAUI	CloneCloud	Cloudlets
Task migration	Yes	Yes	Yes
Virtualization	No	layer: application vm approach: VM migration	layer: hardware vm approach: dynamic VM synthesis
Elastic application division mechanism	method granularity real time choosing of division	thread granularity static choosing of division	No
Bandwidth upgrading and data delivery time	No	No	Yes

among them (i.e. clustering). Furthermore, other researchers try to see the effects of some social factors on the performance. In addition, which type of applications can be handled by MDC and which is the best strategy to split heavy tasks in a way to ensure better performance. Mtibaa et all in [7, 8] investigate for some tracefiles and proof that social offloading make sense compared to random offloading. Moreover, they illustrate for the social factors under some conditions against local execution.

Our work in this research is confined to MDC. Here, we investigate for the effects of social factors in a comprehensive way. Furthermore, we just consider datasets that include our target social factors: friendship and interests. Our goal here is to quantify for the effect of social factors in a way to see which factor give highest gain in term of number of contact, connection duration and intercontact time. We start by cleaning those tracefile from errors, then build a testing application to measure for the gain. We compare and combine all output results too reach common understanding of the most effective factors w.r.t gain. Then we test our module against local execution and a set of proposed offloading algorithm against some predefined task capabilities. Here, we propose that the whole application is already splitted into such tasks, we left the splitting technique as future work. Finally, we test for our propositions in real

simulation environment using the ONE simulator. Moreover, as future work we aim at develop a framework to be run in real devices.

1.4 Context-aware scheme

Each device has its own local context, including: storage capacity, battery level, memory usage, computation performance and network utilization. Context-aware proposed a context within a network of users aimed at fulfills users' satisfaction by monitoring their preferences and provide each with appropriate services. Moreover, context-aware tries to utilize local contexts of communicating devices in order to improve the quality of service (QoS) of the whole system[14].

1.5 Related work

Finding an efficient way to offload heavy tasks to other cooperative devices is still an issue. This is due to the intermittent behavior of the network, and finding an efficient and dynamic way for splitting tasks considering the resource availability of the potential offloadees. In [7]A. Mtibaa et al. introduced a highly collaborative MDC scheme and developed a platform based on real testbeds. They measure the effectiveness of their computation offloading algorithms in extending the lifetime of an MDC. FemtoClouds proposed by K. Habak et al. in [15] as a dynamic and self-configured cloud to leverage the mobile devices computation based on scalable heuristic solution. C. Shi et al. in [16] presented IC-Cloud; a framework for computation offloading considering highly variable quality and intermittent of Internet services. Indeed, in [17] C. Wang et al. tried to investigate contact patterns of the mobility of the communicating devices. While M. Barbera et al. in [18] studied the feasibility of computation offloading and data backups in real mobile scenarios. They consider two types of clones: the off-clone; to support computation offloading purposes, and the back-clone; to be used when restoring user's data and apps needed. Listing of basic requirements for developing a MCC application and design guidelines for better MCC-approach introduced by G. Orsini et al. in[19] and A. Khan et al.in [20]. Furthermore, in [21] B. Zhou et al. proposed an algorithm to deal with offloading decision at runtime on selecting based on the device context. Furthermore, CloudAware proposed by G. Orsini et al. in [22], to deal with changing in context of mobile environments for dynamic code offloading mechanisms. Furthermore, HYCCUPS framework introduced as mobile-to-mobile contextual offloading by RC. Marin et al. in [23], in order to implement on-the-fly contextual offloading model in opportunistic networks.

Other architectures for mobile device clouds proposed by Serendipity[24] and Cirrus [25]. In Cirrus, a mobile device cloud formed based on the spectrum of devices within MDC, also Cirrus and proposes a solution to cyber foraging. This can be done by

considering some powerful computers/sorrugents placed within moving vehicles located in several areas of a building. In other words, Cirrus propose two offloading scenarios for offloading: MDC and cloudlet. While Serendipity is only deal with offloading to mobile device cloud. Serendipity aims to build a testing system to deal with task allocation in MDC in a way to speedup for the task execution and maintain the consumed energy.

The rest of the thesis is organized as follows: Chapter 2 illustrates the core of our thesis which is doing a comprehensive study to investigate for the effects of social aspects. In the same chapter, we manipulate the datasets used and define for metrics and constrains. Chapter 3 tests for different offloading algorithms within a custom app built for this purpose. In the same chapter, we consider an offloading strategy based on the outcome of the investigation study. The last chapter tests for the different offloading algorithms in real simulation environment.

Chapter 2

Social Dataset Investigation

Studying the social behavior between mobile users in MCC field allow us to understand the social communication between users. In addition, it is possible to understand useful encounters for message forwarding in MCC environment. As a result it can be also useful for better designing of efficient offloading algorithms.

Our main purpose of doing this investigation study is to see how mobile devices behave in real environment and how much the social aspects affect the interaction between those devices. This help us in better understanding of the correlation between social parameters and opportunistic meets.

In our study we investigate for the social aspects of two trace files: Sigcomm09[26] and Unical14 [27]. In fact, those datasets are the biggest in terms of number of nodes that include our target social aspects: friendship and interests.

2.1 Dataset Description

We study two real-life human mobility experiments where mobile devices interconnected via Bluetooth link. In each experiment, Bluetooth is used to record for opportunistic contacts among the experimental devices. The datasets are chosen to include different possible environments; from campus to conference. In addition, all of those datasets

contain social information about the participants. The details of the datasets are discussed below.

Sigcomm09. This is a trace of Bluetooth device encounters among a group of users carrying smartphone devices at SIGCOMM 2009 conference, in Barcelona, Spain. The devices perform a device discovery every 120+/-10.24 seconds. Furthermore, each device was initialized with the social profile of the participant that included some basic information such as home city, country and affiliation. In addition, each participant was asked to log on to their Facebook profile in order to include the list of Facebook friends and interests in the social profile. The final trace contains data from 76 devices that show significant activity during the experiment. An opportunistic mobile social application called *MobiClique* was used during the experiment to allow participants to customize their list of friends and interests.

The same dataset include traces of Bluetooth device proximity (interaction among those devices), opportunistic message creation and dissemination, and the social profiles (friends and interests) of the participants for 4 continuous days during the same conference[26, 28].

Unical14. The second dataset we analyze comes from the University of Calabria, Italy. The experiment is based on a Bluetooth with a range of about 10 meters. Each device performs a periodic Bluetooth device discovery every 180 seconds to find out nearby devices. The smartphone's hardware of each device is different, but all of them running Android operating system. Bluetooth device proximity of data collected by an ad-hoc Android application called *SocialBlueConn*. This application was used by 15 students at the same university campus. The dataset includes the social profiles (Facebook friends and self-declared interests) of the participants. The experiment lasted for one week during students' lessons, from January 28, 2014 to February 5, 2014, including only the working days from from 1:00 pm till 7:00 pm [27, 29].

The characteristics of both datasets are summarized in Table 2.1.

Table 2.1: Characteristics of data proximity for traces used in our study

Property	Trace1	Trace2
Dataset name	Sigcomm09	Unical14
Setting	Conference	Campus
Device type	HTC s620 Windows Mobile smartphone	Android mobile devices
Operating System	Windows Mobile	Android
Radio range	~10 m	~10 m
Granularity	120 sec	180 sec
Duration	4 continuous days	7 working days with gaps
# of nodes	76	15

2.2 Study assumptions and methodology

In our study, we deal with our target datasets in raw format. The raw format has possibly some errors or noise. In addition, each dataset has it own way to collect data from mobile users. We work toward unifying the processing of all datasets in one standard format. Then, we clean the datasets from possible errors and noise and define the metrics used through our study. The main goal of doing our investigation study is to see which social factor or combination factors has the big impact in terms of connectivity metrics.

2.2.1 Contributions

Our contributions in this study are threefold: (i) do a sanity check for the number of contacts per hour for the basic datasets. Here, we study the hour-by-hour distribution for the number of contacts in each dataset. This distribution provides us with how many contacts found among devices during each hour and at which hour(s) there is a peak(s), and at which there is a dip(s). (ii) clean and process datasets from noise; we try here to clear noise in each dataset before starting of the study by excluding dip hours from the original datasets. Dip hours are indicators for no activity or little activity; this is manly due to the participants are in rest hours or gradually leave the experiment during the last days. (iii) define our metrics used through the study and

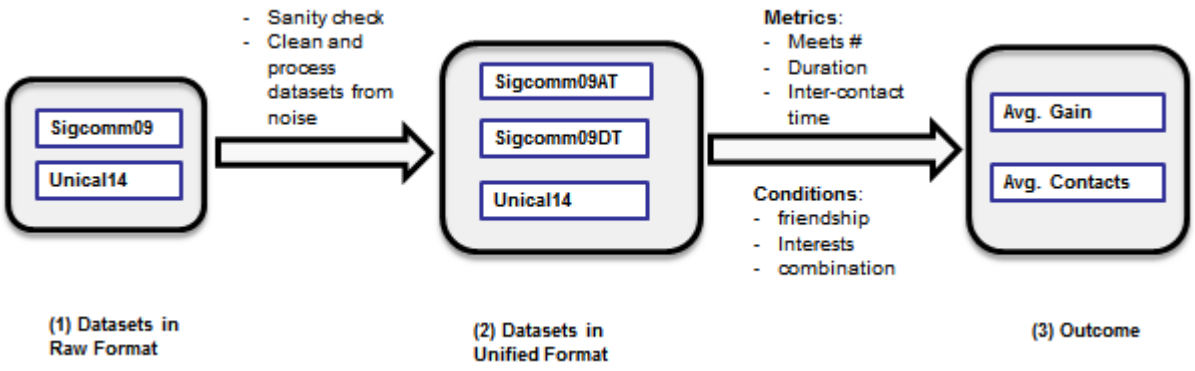

Figure 2.1: Investigation phase process flow

the constraints related to combining social aspects; here, we focus on three metrics: number of contacts, duration of such contact and the intercontact time. We discuss those metrics later in this chapter. Figure 2.1 shows the process flow of our investigation phase.

2.2.2 Trace manipulation

As mentioned earlier, we propose two tracefiles: *Sigcomm09* and *Unical14*. Actually, those datasets are in their raw format. We manipulate the datasets in a way to be useful in understanding and analyzing for the social factors existing in those datasets.

Sigcomm09 Sigcomm09 is a tacefile of 76 nodes communicating together based on friendship and interests shared between a pair of nodes. The raw format for Bluetooth proximity in this dataset is in proximity.csv file. This raw file records for all nearby Bluetooth devices reported by the periodic Bluetooth device discoveries each with 120 seconds. The format of this file is:

timestamp	*user_id*	*seen_user_id*	*device_major_cod*	*device_minor_cod*

, where *timestamp* is the relative time in seconds since the start of the experiment, 17/08/2009 08:00. The user_ids below 100 are the experimental devices, while user_ids >= 100 are external Bluetooth devices seen during the experiment. The *device_major_cod*

and *device_ minor_ cod* correspond to the device's standard Bluetooth Class of Device values. While *user_ id* and *seen_ user_ id* are correspond to the ids of the source node and destination node. Here we exclude for any contact outside the devices ids (i.e. we just consider the contact if both source and destination ids are between 1 and 76) [26].

In our study, we consider each proximity between source and destination nodes as 120 seconds which corresponds to each row in the original proximity.csv file. We combine each successive connections between the same pair as one contact with the combined value as duration of such contact. We did this by building a custom JAVA module, to know exactly each contact duration and to compute the total number of contacts between any pair of nodes. Then we compute for the intercontact time, which is the waiting time for any pair of nodes to be contacted again.

Respect to social factors, the original dataset contains two files representing friendship: *friends1.csv* and *friends2.csv*. The first file represent for the initial friendship graph of the participants based on their Facebook friends. While the second file, represents for the evolution of the friendship graph where users allowed to discover and add other friends upon opportunistic encounters with them.

Similarly to the friendship, the interests found in two files in the original dataset: *interests1.csv* and *interests2.csv*. The first file refers to the initial interest groups of the participants based on their Facebook groups and networks. The list contains also three per-configured common groups for each participant with ids 1,2 and 3, which based on *institute*, *city* and *country* respectively. All of those files have the same format of: | *user_ id* | *user_ id* |; considering the ids of a pair of users have this kind of social relationship.

After that, we build another custom JAVA app to read from all those files in order to compute the following parameters:

- *commonFriends*: number of common (shared) friends between source and destination nodes

- *commonInterests* : number of common interests between source and destination nodes

- *friendship*: if there is a direct friendship between the source and destination nodes (i.e. destination node is a friend of source node).

The new format of Sigcomm09 file is:

srcHost	*dstHost*	*Meets*	*Duration*	*IntercontactTime*	*CommonFriends*	*CommonInterests*	*AreFriends*

Unical14 Unical14 is another tacefile of 15 nodes communicating together based also on friendship and interests shared between a pair of nodes. The raw format for Bluetooth proximity for this dataset is in *Bluetooth_ contacts.txt* file. This raw file records for all nearby Bluetooth devices reported by the periodic Bluetooth device discoveries each with 180 seconds. The format of this file is: | *srcHost* | *dstHost* | *timestamp* |, where the first two fields represent for the ids of the communicating pair. *timestamp* here is the relative time in millieseconds since the start of the experiment, 28/01/2014 13:00[27].

As in the previous dataset, we consider each proximity between source and destination nodes as 180 seconds which corresponds to each row in the proximity file. Then, we combine each successive connections to be as one contact with the combined value as a duration of such contact. To accomplish this, we use the same JAVA module as in the previous dataset in order to compute for the duration, number of contacts and intercontact time between a pair of nodes.

Taking into account the social factors, the original dataset contains a file represents friendship: *Facebook_ friendships.txt*. The same file is in a matrix format with 1's and 0's, where 1 refers to the existing of such relation and 0 not. Respect to common interests, there are 9 files corresponding to a self-declared set of interests. Participants' interests were collected at the beginning of the experiment through an offline questionnaire. The questionnaire contained a list of questions regarding participants' preferences according to some types of activities and hobbies. Each of those files as in

matrix format with 1's and 0's, where the value of 1 represents for the existence of an interest between a pair of nodes. Here, we process the files related to common interests in this dataset in order to compute for the number of interests between a pair of nodes.

After that, we compute *commonFriends*, *commonInterests*, *friendship* values as in the previous dataset. The new format of *Unical14* file is the same as *Sigcomm09* in order to unify the processing of the social factors in the upcoming steps.

Furthermore, in this dataset we find that the minimum number of interests between any pair of nodes is 2, which reflects a high impact of common interests in the same dataset. For that reason and to ensure consistency in our investigation study, we make mapping for the number of interests between the two datasets. So, in *Sigcomm09* the value where there is no interests (*commonInterests*=0) is corresponding to (*commonInterests*=2) in *Unical14* dataset.

2.2.3 Connectivity metrics

In this study we include 2 basic datasets: *Sigcomm09* and *Unical14*. Both of them include friendship and interests as social aspects. Our main goal in this study is to see the effects of those social aspects (friendship and interests) on opportunistic contacts. A *contact*, which is also called a *meet* between two nodes: a and b. The contact occurs when a moves in the proximity of b, so that a is able to send a message to b. The duration of a contact represents how long this condition holds. In addition, the intercontact time is the time elapsed from when b is out of the range of a till it becomes reachable again. All these information are extracted from the trace files and they are measured when a specific constrains C, like C="a is a friend of b", holds. We first define the following quantities:

- *Contacts(C)*: the total number of contacts in the whole trace file satisfying the constrain C.
- *Pairs(C)*: the total number of pairs in the trace file that satisfy C.

- *Durations(C)*: the sum of the whole duration values for all contacts satisfying C.
- *Intercontacts(C)*: sum of the whole intercontact time values for all contacts satisfying C.

From here, we then define the following metrics:

- Average number of contacts between an ordered pair of nodes satisfying C as*:*

$$AvgContact(C) = \frac{Contacts(C)}{Pairs(C)}$$

- Average duration of a contact between an ordered pair of nodes satisfying C as:

$$AvgDuration(C) = \frac{Durations(C)}{Contacts(C)}$$

- Average intercontact time between nodes satisfying C as:

$$AvgIntercontact(C) = \frac{Intercontacts(C)}{Contacts(C)}$$

2.2.4 Effects of social aspects and related conditions

In order to study the effects of social aspects on the above metrics, we define several conditions to be applied on a pair of nodes (i.e. a and b). The first condition is related to friendship. The corresponding condition is: C="b is a friend of a". Intuitively, friend nodes should meet more often than a random pair of nodes. Another aspect of our study is for a pair to have some common friends. The selecting condition is: C="a and b have at least k friends in common". The last condition concerns about common interests. Similarly, the condition is now: C="a and b have at least k interests in common". In other words the condition between a pair of nodes a and b can be defined as follows:

$$C(a,b) = \begin{cases} direct\ friendship \\ commonFriends \geq k,\ k > 0 \\ commonInterests \geq k,\ k > 0 \end{cases}$$

2.3 Study Observations

Before investigating for the effects of social factors on connectivity properties we did a sanity check of the trace files by computing the total number of contacts occurred during each hour. The results are reported in figures 2.2 and 2.3. in Figure 2.2, We can see how for *Sigcomm09* trace a higher number of contacts occurs during daytime period of the first three days whereas for the last day no significant number of meets occurs. For this reason we exclude the last day of *Sigcomm09* from our study as reported in Figure 2.4. We then define a subtrace, hereafter called *Sigcomm09DT* (Day Time) including data from 9:00 am till 6:00 pm of the first three days as illustrated in Figure 2.5. The selection of *Sigcomm09DT* period is based also on the actual workshop sessions of the same conference. The whole tracefile will be referred as *Sigcomm09AT* (All Time). As far as *Unical14* tracefile is concerned, we can see in Figure 2.3 a repeating pattern for the number of contacts during the 7 days of the event. Recall that this dataset is not continuous as it goes from 1:00 pm till 7:00 pm during only working days of the experiment. Furthermore, we noticed that few contacts are present in the last hour of each day due to the end of daily lessons. So, in our study we exclude the last hour of *Unical14* as seen in Figure 2.6.

In addition, to make our investigation study more consistent we exclude any contact in the trace file that satisfies any of the following constraints for intercontact time values:

- Any value for intercontact time that exceeds the tracefile timestamp limit. Each tracefile has defined by starting and ending timestamp, so the intercontact time value cannot exceed the last timestamp of the tracefile.

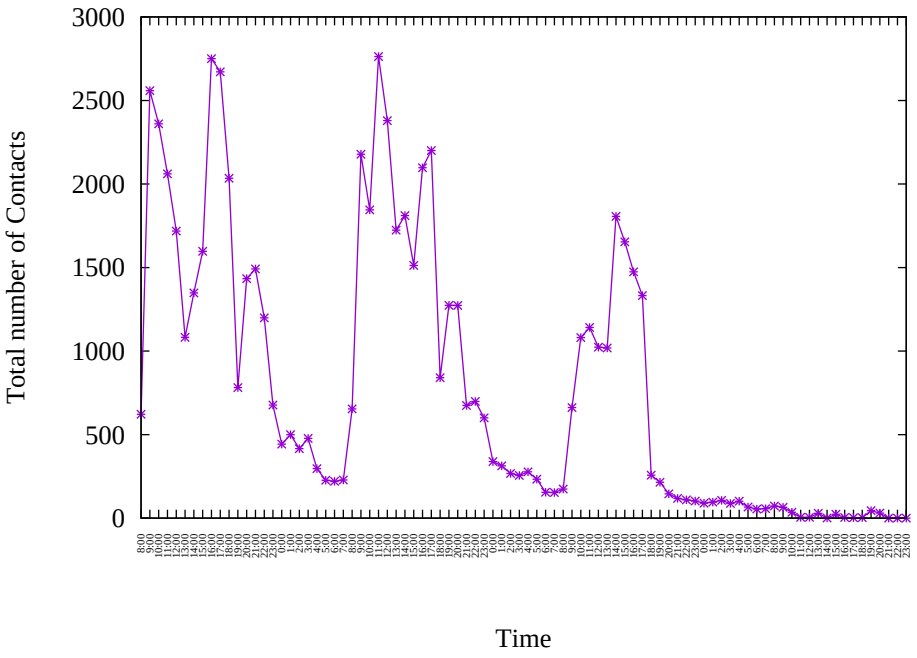

Figure 2.2: Sigcomm09 hour-by-hour contacts distribution

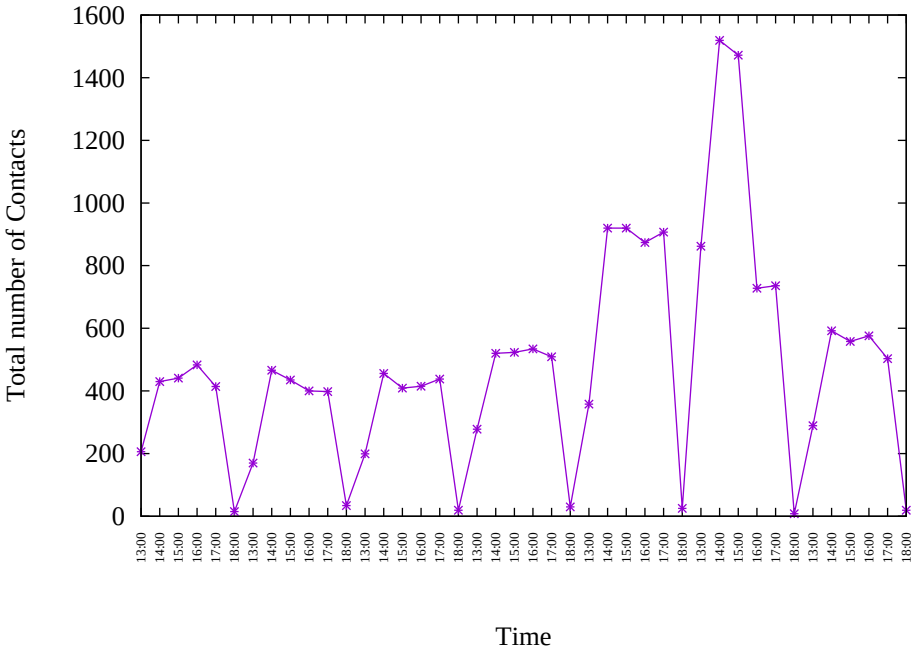

Figure 2.3: Unical14 hour-by-hour contacts distribution

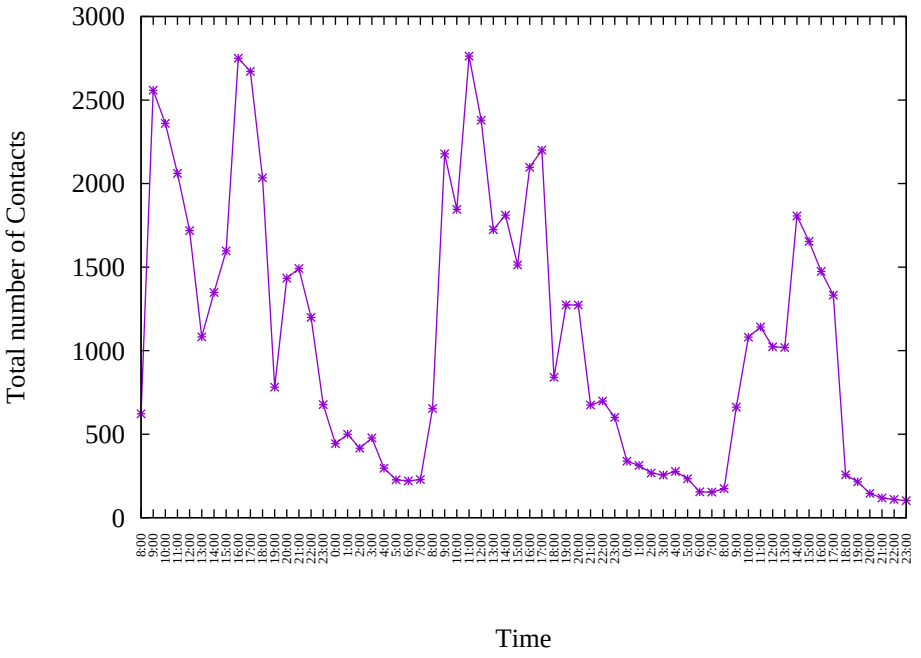

Figure 2.4: Sigcomm09AT hour-by-hour contacts distribution (updated)

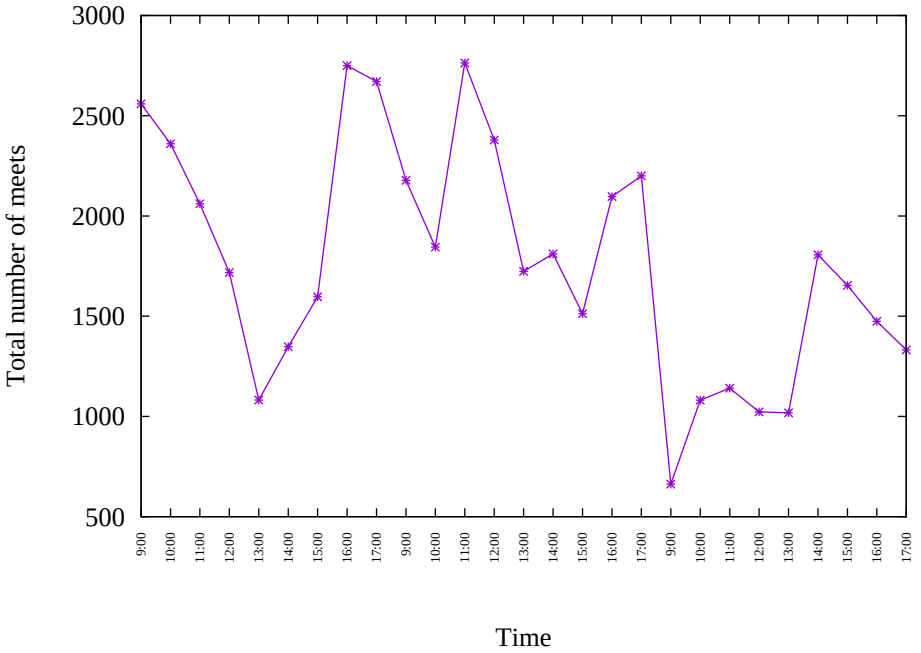

Figure 2.5: Sigcomm09DT hour-by-hour contacts distribution

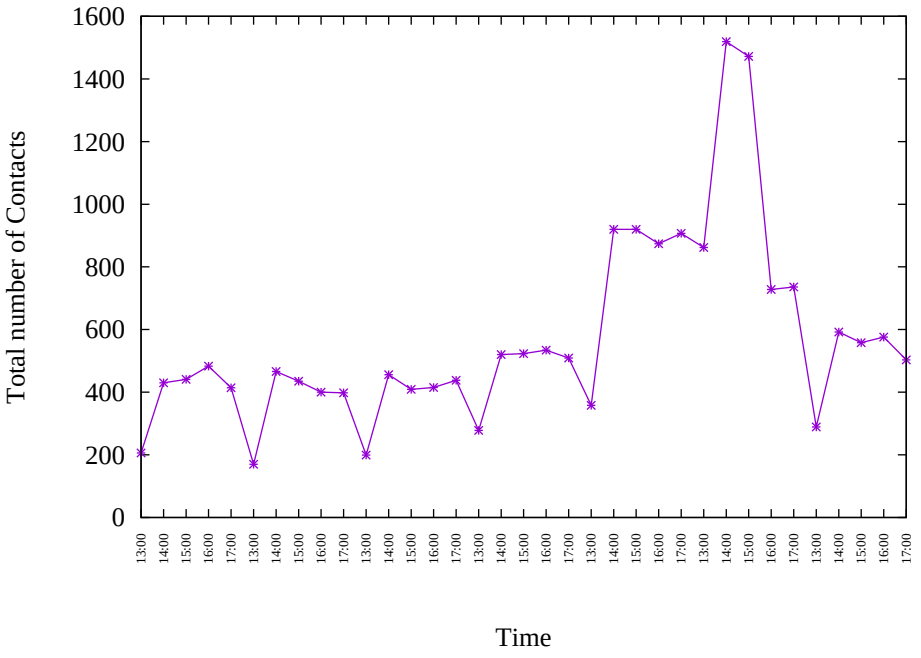

Figure 2.6: Unical14 hour-by-hour contacts distribution (updated)

- Any value for intercontact time that exceeds the current day limit. We identify this constraint to limit the study for the interaction between nodes during the period of each day.

In fact, we try to define a specific set of conditions to be used through our study based on what we mentioned in Section 2.2.4. To achieve this, we define two types of conditions; the first (main) condition concerns of basic social aspects, common friends or common interests, extracted from the trace file. We denote this as *C1*. The other (secondary) condition, denoted as *C2*, is about combining specific number (k) of social aspects with *C1* ; (i.e. combining k interests with common friends or combining k friends with common interests). Moreover, *C2* can include the direct friendship between a pair of nodes, node a and node b. Table 2.2 summarizes the whole set of the conditions for friends in common used for in our investigation study. Similarly, Table 2.3 list for the set of conditions regarding interests in common.

Table 2.2: Description of conditions used for friendship aspect in our study (* for *Unical14*, C2 referred to 4, 6 and 8 interests respectively)

Condition type	Condition description	notation
Main Condition (C1)	nodes a and b have k common friends or more, $k > 0$	C1: Common friends or more
Secondary Condition (C2)	nothing (empty condition)	C2: $\emptyset$
	nodes a and b have at least 2 interests or more*	C2: interests >= 2
	nodes a and b have at least 4 interests or more*	C2: interests >= 4
	nodes a and b have at least 6 interests or more*	C2: interests >= 6
	node b is a friend of node a	C2: direct friendship

Table 2.3: Description of conditions used for friendship aspect in our study

Condition type	Condition description	notation
Main Condition (C1)	nodes a and b have k common interests or more, $k > 0$	C1: Common interests or more
Secondary Condition (C2)	nothing (empty condition)	C2: $\emptyset$
	nodes a and b have at least 2 friends or more	C2: friends >= 2
	nodes a and b have at least 4 friends or more	C2: friends >= 4
	nodes a and b have at least 6 friends or more	C2: friends >= 6
	node b is a friend of node a	C2: direct friendship

2.3.1 Sigcomm09AT study

We start our investigation with the biggest tracefile- *Sigcomm09*, which it includes 76 hosts. This study tries to illustrate the effects of the two social aspects - common friends and common interests - recorded during the conference days. Here, we apply for the constraints as described in Table 2.2 against connectivity metrics.

2.3.1.1 Friendship effect

We found that *Sigcomm09* dataset includes continuous number of common friends up to 13. Figure 2.7 reports for the average number of contacts under two conditions. The first (main) condition denoted as C1="k common friends or more", where k is reported in x-axis. The second (secondary) condition is about common interests and can be formalized as C2="k common interests or more" or C2="direct friendship". Using these two conditions allow us to combine the two fundamental aspect of social factors. Note that the baseline value (condition C1 not satisfied) corresponds to x=0 and no secondary condition at all. We can see how for C2=∅, the average number of contacts increases starting from 6. Adding direct friendship constrain has a positive impact up to 9 common friends, whereas the average number of contacts decreases for higher values. We refer this to lower number of pair of nodes that satisfying this condition which tampers the statistics. While adding "2-Interests or more" or "4-Interests or more" has a dramatic increase on the average number of contacts starting from common friends equal to 9 till 13. Finally, adding number of interests greater than 4 (as illustrated for C2:"interests>=6") result in negative impact. This is due to the lower number of contacts between pairs that satisfying this combination.

In Figure 2.8 we measure for the average intercontact time, here we exclude for any value exceeds the tracefile end timestamp. We noticed that for the baseline value (C2=∅), the average intercontact decreases (positive impact) starting from 8. While adding direct friendship constraint will result in a reasonable positive impact till 9 com-

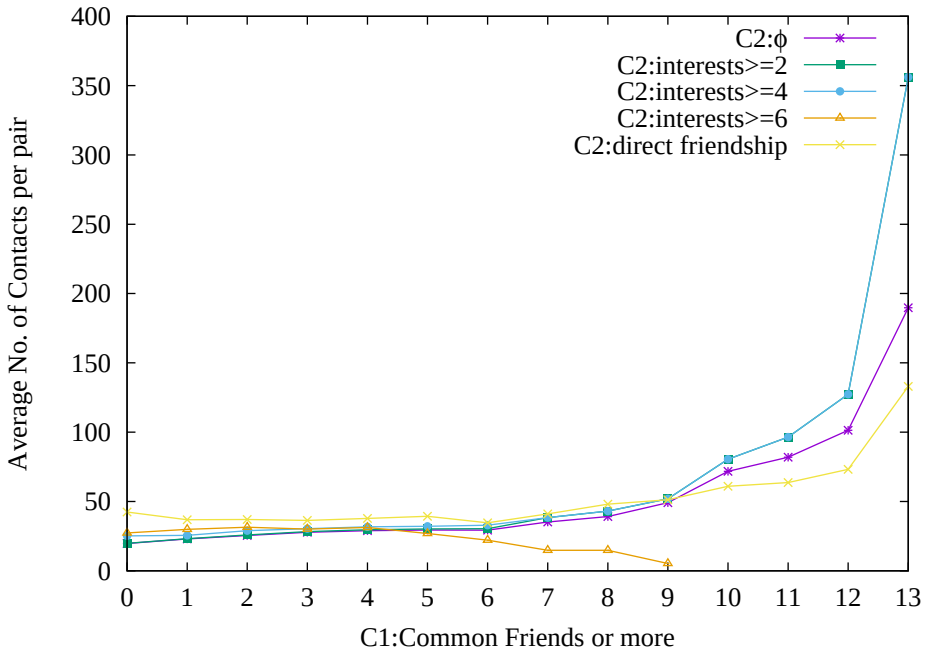

Figure 2.7: Effect for the number of common friends with avg. number of meets for Sigcomm09AT dataset

mon friends then increases (give negative impact) after that. At that point (9 common friends) forward the impact goes equally to "2-Interests or more" or "4-Interests or more" constraints. Moreover, combining more than 4-Interests will result in big negative impact for higher values of common friends as illustrated for C2="interests>=6". This is due to lower number of contacts of pairs that satisfying this combination.

The big image regarding friendship is not clear, so we try to quantify for the connectivity metrics against different social parameters as illustrated in Table 2.4. In the same table we try to see the effect of direct friendship and different numbers of friends in common. Moreover, we try to combine friends in common with direct friendship. From the same table, it is clear the high impact of direct friendship among other social parameters. In other words, in *Sigcomm09AT* we can say that a pair of nodes that are direct friends have the highest number of contacts and lower intercontact time among

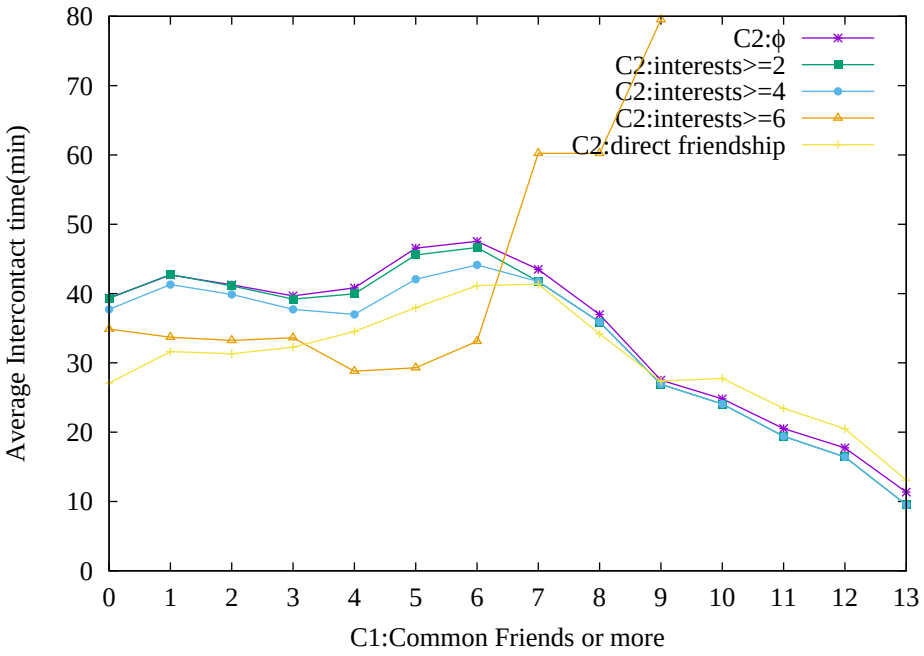

Figure 2.8: Effect for the number of common friends with avg. intercontact time for Sigcomm09AT dataset

other combination related to friendship. So, they are meet more often. Moreover, direct friendship has the highest percentage of contacts respect to the whole tracefile with more than 42%.

Table 2.4: Study Results regarding friends for Sigcomm09AT trace

Connectivity metric	Social parameter							
	None	F	2-Fri	4-Fri	6-Fri	F & 2-Fri	F & 4-Fri	F & 6-Fri
Avg Contacts (per pair)	19.7	**42.4**	24.3	26.2	29	37	37.8	38.2
Avg Intercontact (min)	39.4	**27.1**	29.7	30.1	31	31.3	34.5	34.8
Percentage of whole contacts	100%	**42.3%**	33.8%	31.5%	29%	26.9%	21.6%	16.3%

2.3.1.2 Interests effect

Respect to interests, we found that *Sigcomm09* file includes continuous number of common interests till 10. Here we define two conditions related to common interests; main condition denoted as C1="k common interests or more", where k is reported in

x axis. While the secondary condition is about common friends and can be formalized as C2="k common friends or more" or C2="direct friendship". Figure 2.9 reports for the average number of contacts under C1 and C2 conditions. For C2=∅, we can see how the average number of contacts increases starting from 3 upwards. Adding direct friendship constrain gives a positive impact across any number of common interests. While adding "4-Friends or more" has a dramatic increase for the average number of contacts starting from common friends equal to 6. This refers to the highest number of contacts for pairs satisfying the combination. Finally, adding number of friends greater than 4 (as illustrated for C2:"friends>=6") result in negative impact since the number of contacts for pairs satisfying this condition will be lower.

In Figure 2.10, we notice that for C2=∅ that the average intercontact time decreases (positive impact) starting from 6 common friends. While adding direct friendship constraint will result in a positive impact over any number of common interests. Indeed, adding "2-Friends or more" as constraint gives reasonable positive impact starting from 6. While for "4-Friends or more" constraint, the impact is higher starting from 4 common interests. Those observations refers to high number of contacts for pairs that satisfying those conditions. However, combining more than 4-Friends will result in a high negative impact for the average intercontact time as seen for "C2:6-Friends or more". This is due to lower number of contacts of pairs that satisfying this combination.

The whole picture regarding interests is not clear, so we try to quantify for the connectivity metrics against different social parameters as illustrated in Table 2.5. In the same table we try to see the effect of direct friendship and different numbers of interests in common. From the same table, it is clear the high impact of combining direct friendship with at least 4 interests. In other words, in *Sigcomm09AT* we can say that a pair of nodes that are direct friends and share at least 4 interests have the

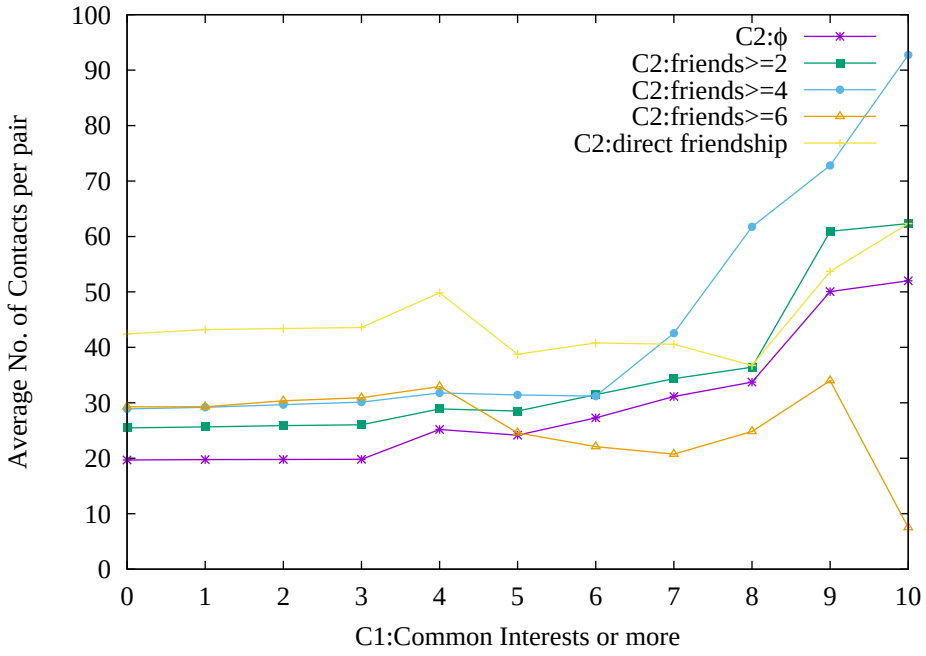

Figure 2.9: Effect for the number of common interests with avg. number of meets for Sigcomm09AT dataset

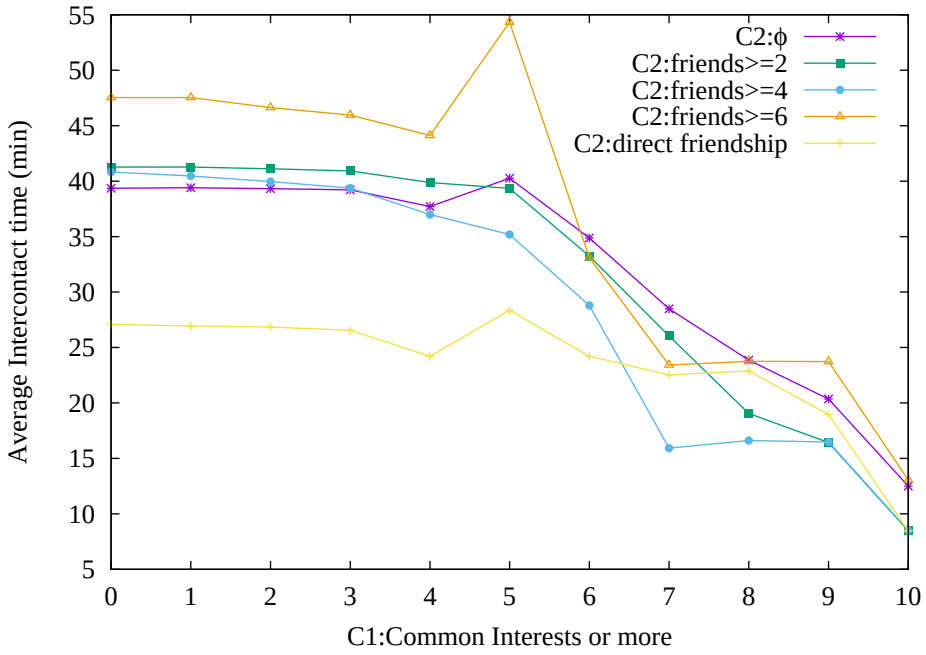

Figure 2.10: Effect for the number of common interests with avg. intercontact time for Sigcomm09AT dataset

highest number of contacts and lower intercontact time among other social parameters. However, a pair of nodes with at least 4 common interests have the highest number of contacts (about 69% among all contacts).

Table 2.5: Study Results regarding interests for Sigcomm09AT trace

Connectivity metric	Social parameter							
	None	F	2-Int	4-Int	6-Int	F & 2-Int	F & 4-Int	F & 6-Int
Avg Contacts(per pair)	19.7	42.4	22.5	25.2	21.3	43.4	**49.8**	41.8
Avg Intercontact(min)	39.4	27.1	39.7	37.7	40.2	26.8	**24.2**	27.9
Percentage of whole contacts	100%	42.3%	61.1%	**68.9%**	57%	34.6%	30.2%	27.8%

2.3.2 Sigcomm09DT study

As mentioned earlier, *Sigcomm09DT* is a sub-tracefile from the original *Sigcomm09* file. It includes data recorded during the workshop sessions of the same conference from 9:00 am till 6:00 pm. Here, we investigate for the main social factors: common friends and common interests.

2.3.2.1 Friendship effect

The average number of contacts is reported in Figure 2.11. The x-axis concerns about the main condition (C1) which is related to k common interests or more. The secondary condition (C2) is about combining some common interests or direct friendship. From the same figure, we notice that the average number of contacts increases starting from 6 at the baseline where C2=∅. By adding C2="direct friendship", the average number of contacts increases till 9 common interests and decreases for higher values. This behavior refers to the lower number of pairs of nodes that satisfying this condition. Furthermore, by adding "2-Interests or more" or "4-Interests or more" gives a dramatic increase on the average number of contacts starting from common friends equal to 9. Adding number of interests over 4, as illustrated in C2:"interests>=6", result in negative impact for the

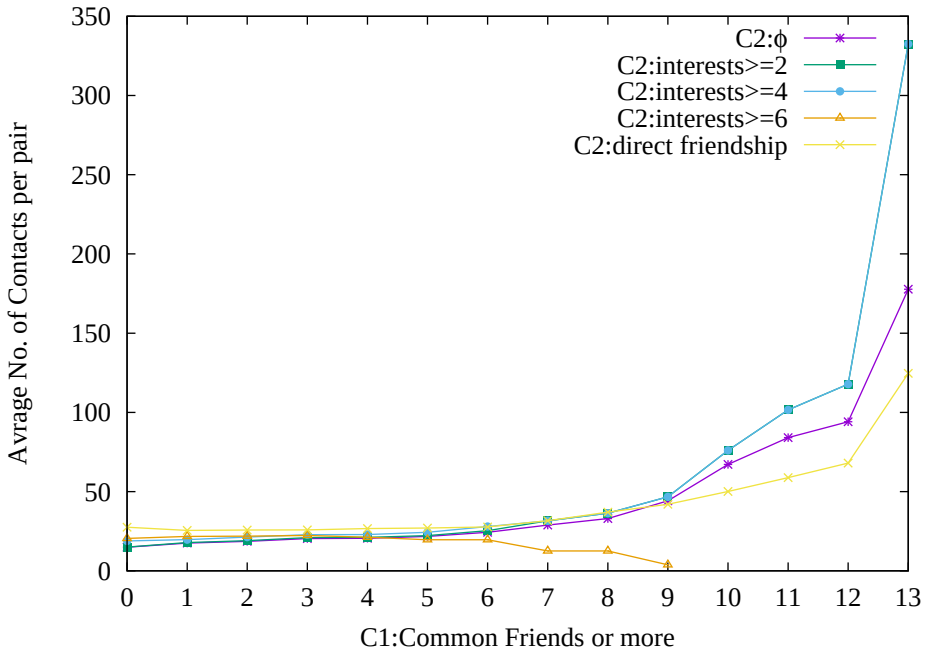

Figure 2.11: Effect for the number of common friends with avg. number of meets for Sigcomm09DT dataset

number of contact. This is due to the lower number of contacts between pairs that satisfying this combination.

Figure 2.12 illustrates for the average intercontact time. We noticed that for the baseline value (C2=∅), the average intercontact decreases (positive impact) starting from 8. While adding direct friendship constraint, will result in a reasonable positive impact till 9 common friends then increases (give negative impact) after that. At that point (9 common friends) forward, the impact goes equally to "2-Interests or more" or "4-Interests or more" constraints. Moreover, combining more than 4-Interests will result in big negative impact for higher values of common friends as illustrated for C2="interests>=6". This is due to lower number of contacts of pairs that satisfying this combination.

Table 2.6 presents the quantifying values regarding *Sigcomm09DT* friends Here,

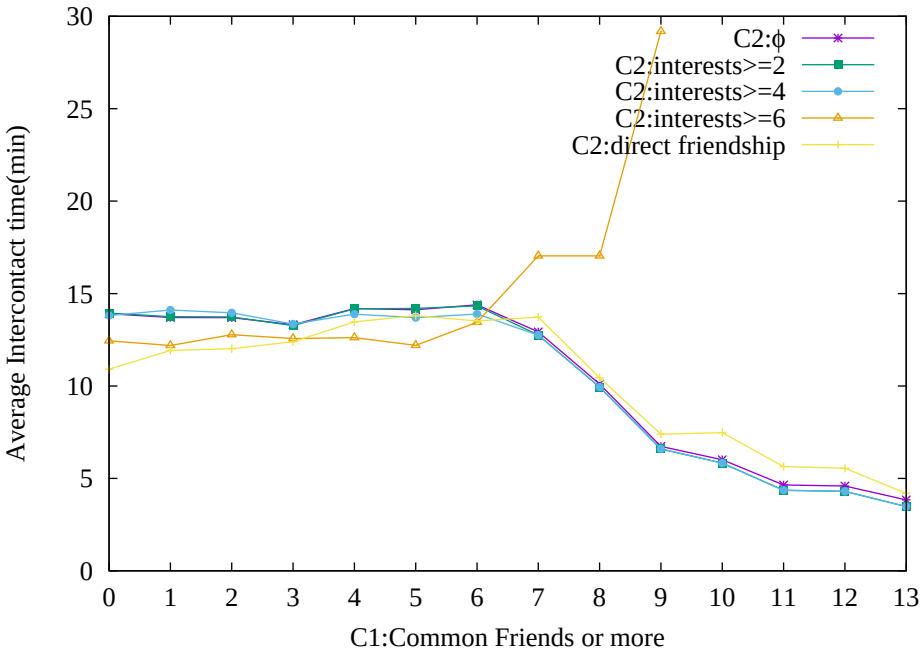

Figure 2.12: Effect for the number of common friends with avg. intercontact time for Sigcomm09DT dataset

we can get the same thing as in Table 2.4, but the percentage of direct friendship respect to the whole dataset is a little bit higher. Moreover, the intercontact time is decreased 3 times than *Sigcomm09AT*. This supports our assumption regarding derive *Sigcomm09DT* from its original dataset.

Table 2.6: Study Results regarding friends for Sigcomm09DT trace

Connectivity metric	Social parameter							
	None	F	2-Fri	4-Fri	6-Fri	F & 2-Fri	F & 4-Fri	F & 6-Fri
Avg Contacts (per pair)	14.9	**27.5**	24.7	26.4	25.3	25.7	26.6	26
Avg Intercontact (min)	13.9	**10.9**	12.7	13.9	13.6	12	13.7	13.1
Percentage of whole contacts	100%	**42.8%**	33%	29.6%	26%	22.3%	18.9%	17.6%

2.3.2.2 Interests effect

Figure 2.13 reports for common interests under C1 and C2 conditions. Referring to C2=∅, the average contacts start to increase from 4 till 7 common interests. The

same figure illustrates regular increase until 4 common interests across other type of combinations. Then those combinations go in zig-zag pattern (up and down). We refer this observation to the number of contacts satisfying a combination.

The average intercontact time is reported in Figure 2.14. The same figure shows that for C2=∅, the average intercontact time decreases (positive impact) starting from 6 common friends. Moreover, adding direct friendship constraint will result in a positive impact over any number of common interests respect to the baseline. Furthermore, adding "2-Friends or more" as constraint gives reasonable positive impact at common friends equals to 10. While for "4-Friends or more" constraint, the impact is higher starting from 6 common interests. Those observations refer to a high number of contacts for pairs that satisfying those conditions. However, combining more than 4-Friends will result in a high negative impact for the average intercontact time as seen for "C2: 6-Friends or more". This is due to very limited number of pairs that satisfying this condition.

Table 2.7 shows the quantifying values regarding *Sigcomm09DT* interests. Here, we can get the same thing as in Table 2.5, but the intercontact time is decreased more than 2 times than *Sigcomm09AT*. This supports our assumption regarding derive *Sigcomm09DT* from its original dataset.

Table 2.7: Study Results regarding interests for Sigcomm09DT trace

Connectivity metric	Social parameter							
	None	F	2-Int	4-Int	6-Int	F & 2-Int	F & 4-Int	F & 6-Int
Avg Contacts(per pair)	14.9	27.5	17.4	18.8	19.3	28.1	**31.6**	30.8
Avg Intercontact(min)	13.9	10.9	13.5	13.8	13.3	11	**10.8**	11.3
Percentage of whole contact	100%	42.8%	51.3%	**55.6%**	47%	33.1%	28.2%	24.8%

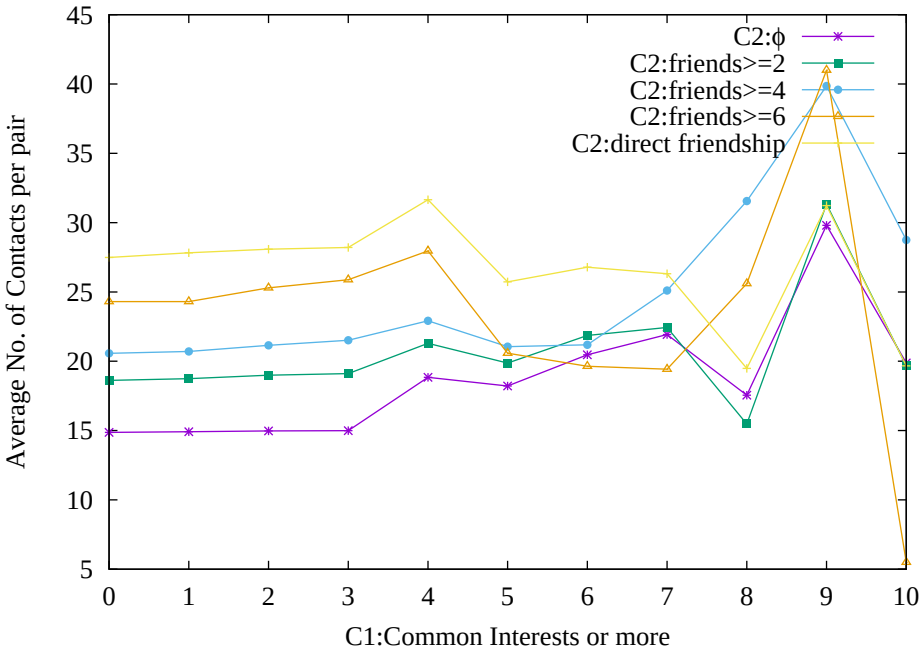

Figure 2.13: Effect for the number of common interests with avg. number of meets for Sigcomm09DT dataset

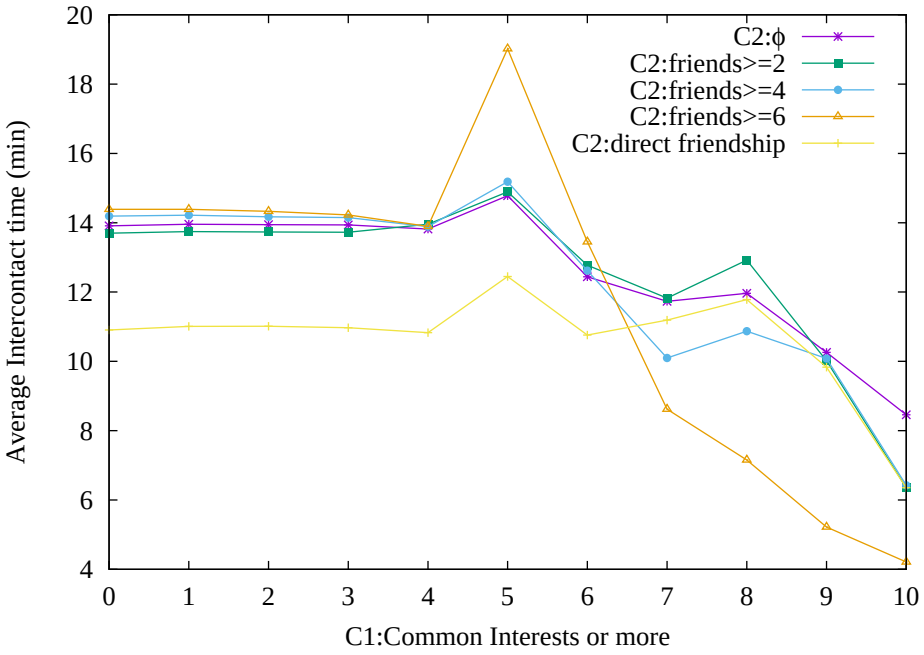

Figure 2.14: Effect for the number of common interests with avg. intercontact time for Sigcomm09DT dataset

2.3.3 Unical14

For Unical14, the experiment lasted for 7 working days (in-continuous days) during class lessons from 1:00 pm till 6:00 pm. In this study. Here, we investigate for the same social parameters: friendship and interests against metrics defined in Section 2.2.3.

2.3.3.1 Friendship Effect

In *Unical14*, we find that the continuous number of common friends is up to 8. Furthermore, the minimum number of interests between any random pair is 2 as mentioned earlier. For that reason and to ensure the consistency for processing this dataset, we make mapping for the number of interests before combining them with common friends. So, both datasets have equivalent number of interests as follows: 4, 6 and 8 interests in *Unical14* dataset are equivalent to 2, 4 and 6 interests in *Sigcomm09* dataset. Figure 2.15 illustrates the average number of contacts under C1 and C2 constraints. In the baseline where C2=$\varnothing$, we can see how the average number of contacts increases until 7 common friends. Indeed, adding direct friendship constrain has a positive impact up to 5 common friends then it becomes same as baseline. Whereas adding any number of interests has positive impact, the only exceptions are for common friends equal to 4 and common friends equal to 8. Those exceptions are due to the lower number of contacts satisfying this kind of combination.

In Figure 2.16 we illustrate for the average intercontact time against number of common friends. Here, we clearly see the up-down plots of Figure 2.15. Figure 2.16 shows that at baseline (C2=$\varnothing$), there is a positive impact (decreasing value) for the average intercontact time. The same behavior noticed for combining any number of interests, just an exceptions in the case of 4 and 8 common friends. This is due to lower values that satisfying those kind of combinations. Moreover, combining direct friendship has positive impact over baseline until common friends equal to 5.

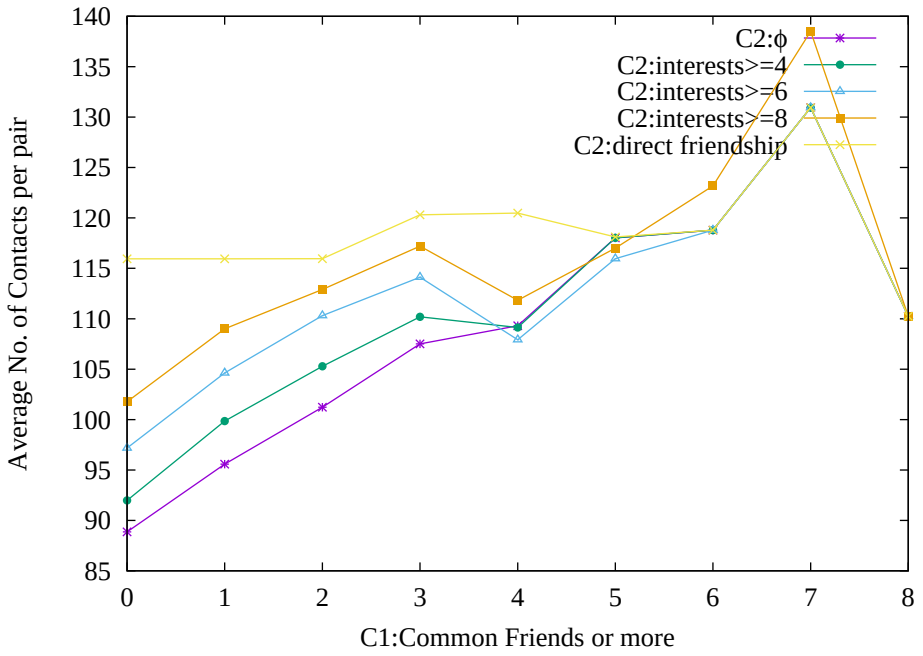

Figure 2.15: Effect of the number of common friends with avg. number of meets for Unical14 dataset

Table 2.8 presents for the quantifying values regarding *Unical14* friends. Here, we can get the same thing as in Table 2.4, but the percentage of direct friendship respect to the whole dataset is lower. This indicates that the direct friendship factor is around a third of the whole contact in the dataset.

Table 2.8: Study Results regarding friends for Unical14 trace

Connectivity metric	Social parameter							
	All	F	2-Fri	4-Fri	6-Fri	F & 2-Fri	F & 4-Fri	F & 6-Fri
Average Contacts (per pair)	88.9	**125.9**	112.4	116.9	119.7	117	120.5	121.3
Avg Intercontact (min)	15.7	**12.8**	14.1	13.8	13.6	13.2	13	13
Percentage of whole contacts	100%	**31.9%**	24.9%	21.7%	19.8%	18.2%	16.7%	12.4%

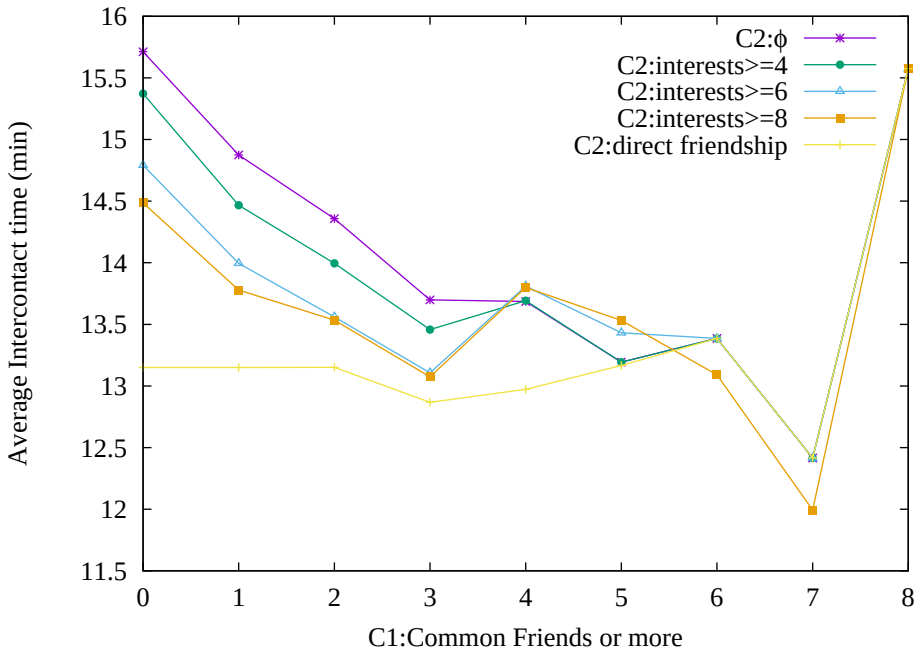

Figure 2.16: Effect of the number of common friends with avg. intercontact time for Unical14 dataset

2.3.3.2 Interests Effect

Here, we find that the continuous number of common interests is from 2 up to 17. Figure 2.17 reports for the average number of contacts under C1 and C2. We can see for C2=∅ (baseline), that the average number of contacts increases over common friends until 9. Moreover, we notice that the highest impact goes to combining 6 friends and for direct friendship respectively. Furthermore, other combinations have positive impact respect to the baseline.

Figure 2.18 illustrates for the average intercontact time, we notice almost the up-down plots for Figure 2.17. For C2=∅, we see the decreasing behavior (positive impact) over number of common interests until 9. Moreover, adding any number of friends has positive impact (lowest values for intercontact time). The highest impact goes to adding 6 friends. Furthermore, adding direct friendship results in a reasonable positive impact.

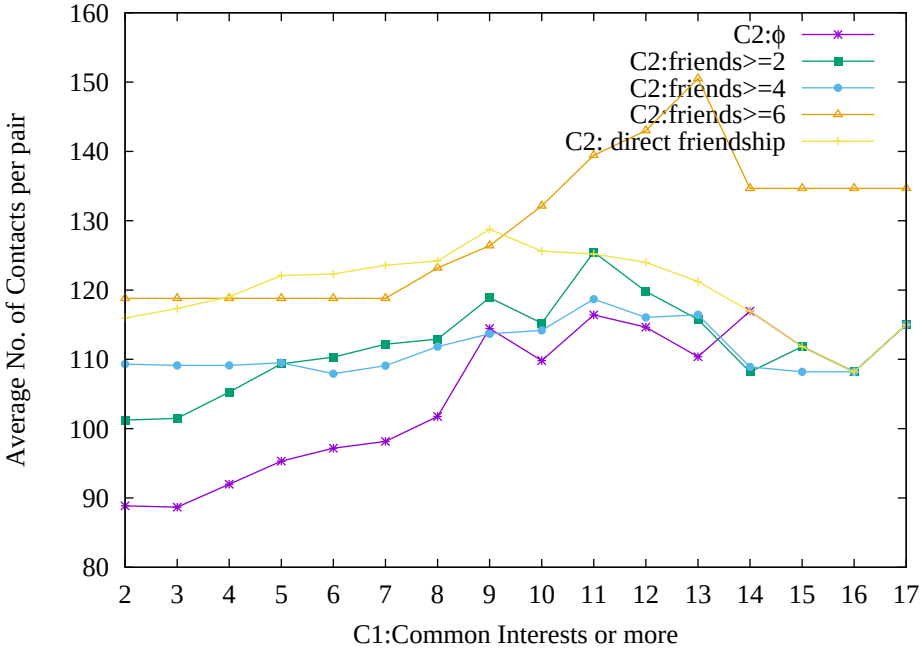

Figure 2.17: Effect of the number of common interests with avg. number of meets for Unical14 dataset

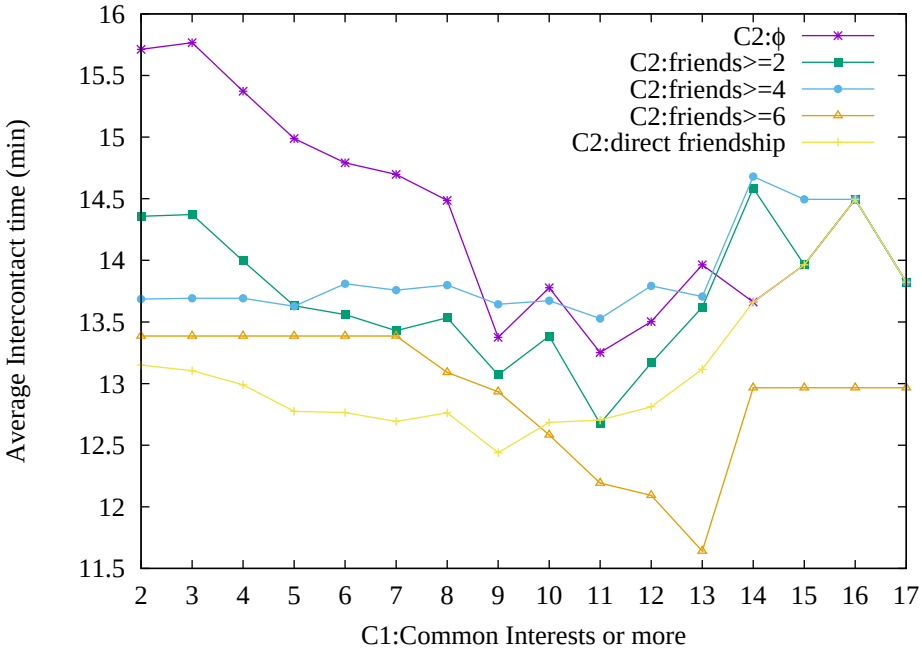

Figure 2.18: Effect for the number of common interests with avg. intercontact time for Unical14 dataset

Table 2.9 presents the quantifying values regarding *Unical14* interests. Here, we can get the same thing as in Table 2.5 respect to the percentage of a pair of nodes sharing at least 4 common interests. The same factor gives 63% as a percentage of contacts respect to the whole dataset.

Table 2.9: Study Results regarding interests for Unical14 trace

Connectivity metric	Social metric							
	All	F	2-Int	4-Int	6-Int	F & 2-Int	F & 4-Int	F & 6-Int
Avg Contacts(per pair)	88.9	115.9	93.1	97.2	88.9	119	**122.3**	112.4
Avg Intercontact (min)	15.7	13.2	16.5	14.8	18.6	13	**12.8**	14.7
Percentage of whole contact	100%	31.9%	56.3%	**63%**	50.3%	29.7%	26.4%	23.7%

We noticed during our study that the average contact duration for all datasets are the same value specified as granularity (as described in Table 2.1). In fact, there are some contacts last for long time (i.e. more than 120 seconds for *Sigcomm09* and 180 seconds for *Unical14*). However, the majority of those contacts are just equal to the granularity value which explains why the average contact duration is the same.

In the next section, we try to quantify for the observations stated here and to elaborate about them. The complete picture of which social factor or combination of social factors is not clear yet. So, we aim at arriving for general conclusion about this by doing this kind of quantifying.

2.4 Results and Discussion

In this section, we try to quantify for the observations described earlier. In fact, we limit our work here until specific number of social factors. So, we just include common friends and common interests till the value of 6. This is due to the lower number of pairs in the datasets after that value. Here, we aim at getting better knowledge of the best possible combination of the social aspects. This will help us in designing an efficient

offloading algorithm that can be used to extend the lifetime of the MDC. For the best of our knowledge, we are the first that doing this kind of study in a comprehensive way.

2.4.1 Quantifying

Here, we try to specify the exact values about which social factors or combination of social factors is the best in terms of average number of contact and average gain. Average gain here related to connectivity metrics defined in Section 2.2.3. For that, we use 3 values for percentile to get the score of the dataset observations:

- P10 (10th percentile): is a value (or score) in which 10% of the observations may be found below this score.
- P50 (50th percentile) or median: is another score where half of the observations may fall below that value.
- P90 (90th percentile): this score indicates that 90% of the observations may take place under this point.

Furthermore, we define the following terms that will be used to get the outcome:

- Gain: is the value that we can get over the baseline value. This value is mostly expressed as percentage value. However, the gain can be positive (earning) or negative (loss). In other words, we can define the gain as follows:

$$Gain = \frac{(newVal - baseVal)}{baseVal} * 100\%$$

Moreover, we can define the average gain as follows:

$$AvgGain = \frac{\sum Gain}{GainValNum}$$

GainValNum here is the total number of gain values.

In fact, AvgGain is computed based on the value of median (P50) over each strategy as we will see later. Median is resilient to extremely large or small values. So, it is

considered as a good descriptor for the outcome. Consequently, the higher the value for AvgGain, the better the strategy to follow.

- Rank: is a value giving if a specific condition or state occurs. In our study, we give the value of 1 if this state has been satisfied. Furthermore, we compute the total rank over each strategy as shown later. So, the strategy that has higher total of ranks will be the best.

We apply the terms above on both common friends and common interests. In Table 2.10, we try to quantify for common friends (condition C1) against different possible combinations (condition C2). The same table provides for the percentile values (P10, P50 and P90) over the all datasets used in our study. We include here the basic connectivity metrics: average number of contacts and average intercontact time. However, we exclude the duration of the contact because it remains almost the same as described earlier. Moreover, we compute for the average gain for each strategy based on the no-combination (C2=$\varnothing$). In the same table, we highlight in bold-font and we give a rank of 1 for the highest values of median (P50) . Then, we compute for the average gain and count for the total rank as illustrated before.

Moreover, Table 2.10 indicates that C2="direct friendship" combination gives highest rank of 5 out of 6 with an average gain of 10.8%. Moreover, combining 4-Interests and 2-Interests give lower values for average gain of 3.8% and 2% respectively. On the other hand, combining 6-interest give negative gain.

In Table 2.11 we try to quantify for common interests as main condition (C1) with different combinations (C2) as described earlier. Here, we also apply the same criteria for computing the average gain and getting the rank value as in table 2.10. Furthermore, Table 2.11 indicates that the highest gain of 28.5 goes again to direct friendship with highest rank of 5 out of 6. The same table illustrates that adding any number of friends will give positive gain. In the same table, we show that combining 6-friends will give a gain of 8.5% as the second highest gain.

Table 2.10: Summary of our investigation study regarding friendship respect to No-Combination strategy, (P10: 10th percentile, P50: 50th percentile, P90: 90th percentile)

Metric type	Dataset	No Combination	Direct Friendship	2-interests or more*	4-interests or more*	6-interests or more*
Avg. Contacts per pair (#) {P10, P50, P90}	Sigcomm09-AT	{23.7, 32.3, 95.5}	**{36.5, 41.8, 70.3} {53.8%, 29.3%, -26.4%}**	{24, 34.3, 118} {1.2%, 6.4%, 23.6%}	{26.6, 35.6, 118} {11.9%, 10.3%, 23.6%}	{13.9, 27.1, 31.2} {-41.5%, -16.1%, -67.3%}
	Sigcomm09-DT	{17.9, 26.6, 91.1}	{25.8, 29.7, 65.3} {44.3%, 11.8%, -28.4%}	{18.2, 28.4, 113} {1.7%, 7%, 24%}	**{20.2, 29.8, 113} {13.1%, 12.1%, 24%}**	{11.7, 20, 21.9} {-34.5%, -24.5%, -76%}
	Unical14	{94.2, 109.3, 121.2}	**{114.8, 118.1, 122.6} {21.8%, 8.1%, 1.1%}**	{98.3, 110.2, 121.2} {4.3%, 0.8%, 0%}	{103.1, 110.3, 121.2} {9.5%, 0.9%, 0%}	{107.6, 112.9, 126.2} {14.1%, 3.3%, 4.1%}
Avg. Intercontact time (min) {P10, P50, P90}	Sigcomm09-AT	{18.6, 39.5, 45.6}	**{21.4, 31.5, 40.2} {-15.1%, 20.4%, 11.9%}**	{17.3, 39.3, 44.7} {6.8%, 0.6%, 2%}	{17.3, 37.3, 42} {6.8%, 5.5%, 8%}	{29.2, 33.7, 62.2} {-57.4%, 14.8%, -36.2%}
	Sigcomm09-DT	{4.6, 13.1, 14.2}	**{5.5761, 11.4133, 13.6647} {-21.1%, 13%, 3.6%}**	{4.3, 13, 14.2} {6.2%, 0.8%, -0.1%}	{4.3, 13, 13.9} {6.2%, 0.5%, 1.6%}	{12.2, 12.7, 18.3} {-164.9%, 3.2%, -28.8%}
	Unical14	{13, 13.7, 15.6}	**{12.8, 13.2, 13.8} {2%, 4%, 11.4%}**	{13, 13.7, 15.4} {0%, 0%, 1.2%}	{13, 13.6, 14.9} {0.5%, 1%, 4.2%}	{12.9, 13.5, 14.7} {1.4%, 1.2%, 5.8%}
	Total (out of 6)	**-**	**5**	**0**	1	**0**
	Avg gain (%)	**-**	**10.8%**	**2%**	**3.8%**	**-2.3%**

Table 2.11: Summary of our investigation study regarding interests respect to No-Combination strategy

Metric type	Dataset	No Combination	Direct Friendship	2-friends or more	4-friends or more	6-friends or more
Average Contacts per pair (#) {P10, P50, P90}	Sigcomm09-AT	{19.8, 25.2, 50.1}	**{38.7, 43.2, 53.7} {96.1%, 71.5%, 7.3%}**	{25.7, 28.9, 60.9} {29.8%, 14.7%, 21.7%}	{29.2, 31.4, 72.8} {47.7%, 24.8%, 45.4%}	{20.8, 29.3, 32.9} {5%, 16.3%, -34.2%}
	Sigcomm09-DT	{14.9, 18.2, 21.9}	**{19.7, 27.5, 31.2} {31.9%, 50.9%, 42.5%}**	{18.6, 19.7, 22.4} {24.9%, 8%, 2.4%}	{20.7, 21.5, 31.6} {38.8%, 18.1%, 43.9%}	{19.4, 24.3, 28} {30.3%, 33.4%, 27.5%}
	Unical14	{90.4, 109, 115.7}	{113.4, 121.7, 125.4} {25.4%, 11.6%, 8.4%}	{103.4, 112, 119.3} {14.3%, 2.8%, 3.1%}	{108.2, 109.4, 116.2} {19.7%, 0.4%, 0.5%}	**{118.8, 129.3, 141.2} {31.4%, 18.6%, 22%}**
Average Intercontact time (min) {P10, P50, P90}	Sigcomm09-AT	{20.3, 37.7, 39.4}	**{19, 24.2, 27.1} {6.7%, 35.8%, 31.3%}**	{16.4, 39.3, 41.3} {19.2%, -4.3%, -4.7%}	{15.9, 35.2, 40.5} {21.7%, 6.7%, -2.7%}	{23.4, 44.1, 47.5} {-15.1%, -17%, -20.6%}
	Sigcomm09-DT	{10.2609, 13.8159, 13.9581}	**{9.8, 11, 11.8} {4.2%, 20.6%, 15.6%}**	{10, 13.7, 14} {2.4%, 0.9%, 0%}	{10.1, 13.9, 14.2} {1.7%, -0.5%, -1.8%}	{5.2, 13.9, 14.4} {49.1%, -0.6%, -3.1%}
	Unical14	{13.4, 14.2, 15.5}	**{12.7, 12.9, 13.9} {5.584%, 9.312%, 10.611%}**	{13.1, 13.6, 14.4} {2.4%, 4.2%, 7.1%}	{13.6, 13.7, 14.5} {-1.5%, 3.5%, 6.7%}	{12.1, 13, 13.4} {9.6%, 8.9%, 13.9%}
	Total (out of 5)	**-**	**5**	**0**	**0**	**1**
	Avg gain	**-**	**28.5%**	**3.7%**	**7.6%**	**8.5%**

After that, we take the best strategy in terms of total gain (i.e. direct friendship) and try to combine it with all possible values for common interests and common friends. Moreover, we include 4-interests strategy because it has the highest value for number of contacts. Here, we compute the gain respect to the total number of contacts to see the which strategy is common in all datasets. Moreover, we follow the same ranking strategy as described earlier. Furthermore, we define the average contacts of a combination C as:

$$AvgContacts(C) = \frac{Contacts(C)}{TotalContacts} * 100\%$$

where *Contacts(C)* is the number of contacts under a combination C and *TotalContacts* represent the total number of contacts in each dataset.

The final outcome of our study is reported in Table 2.12. The same table illustrates that the highest gain is for combining direct friendship with 4-Interest. This combination has full total rank with 63.8% as average gain. However, the highest average of contacts goes to 4-Interests constraint with 62.5% respect to the total number of contact in the whole datasets.

In a nutshell, we investigate for the social aspects (i.e. friendship and interests) across the biggest datasets that have those social aspects. We manipulate those datasets in order to clean them from errors and unify their format for processing. Then, we point out for the observations from those datasets. After that, we quantify for those observations in order to get better understanding of which social factors or combination of factors is the best in terms of number of contacts and gain. The outcome of our investigation study indicates that: (i) a social factor with at least common interests has the highest number of contacts, and (ii) combine direct friendship with at least 4 common interests gives the highest gain across connectivity metrics. We think that utilizing those results in an offloading algorithm will be useful and give efficient performance.

Table 2.12: Results regarding all contacts in different datasets of our investigation study

measure	Dataset	All contacts	Direct Friendship (**F**)	4-Interests or more (**4-Int**)	**F** + 2-Interests or more (**2-Int**)	**F**+ **4-Int**	**F**+ 2-Friends or more (**2-Fri**)	**F**+ 4-Friends or more (**4-Fri**)	**4-Int** + **4-Fri** + **F**
Average Contacts (per pair)	Sigcomm09-AT	19.7	42.4 (115.4%)	25.2 (27.9%)	43.4 (120.4%)	**49.8 (153%)**	37 (88%)	37.8 (92.1%)	39.3 (99.7%)
	Sigcomm09-DT	14.9	27.5 (84.9%)	18.8 (26.7%)	28.1 (88.9%)	**31.6 (112.9%)**	25.7 (73.1%)	26.6 (79.3%)	27.4 (84%)
	Unical14	88.9	115.9 (30.5%)	97.2 (9.4%)	119 (33.9%)	**122.3 (37.6%)**	116 (30.5%)	120.5 (35.6%)	118.8 (33.6%)
Average Intercontact time (min)	Sigcomm09-AT	39.4	27.1 (31.2%)	37.7 (4.18%)	26.8 (31.8%)	**24.2 (38.5%)**	31.3 (20.5%)	34.5 (12.3%)	31.2 (20.8%)
	Sigcomm09-DT	13.9	10.9 (21.6%)	13.8 (0.67%)	11 (20.8%)	**10.8 (22.2%)**	12 (13.6%)	13.7 (3.2%)	13.5 (3.1%)
	Unical14	15.7	13.2 (16.3%)	14.8 (5.9%)	13 (17.3%)	**12.8 (18.8%)**	13.2 (16.3%)	13 (17.4%)	13.2 (16.2%)
	Total (out of 6)	**-**	**0**	**0**	**0**	**6**	**0**	**0**	**0**
	Avg gain (%)	**-**	**50%**	**12.4%**	**52.2%**	**63.8%**	**40.3%**	**40%**	**42.9%**
	Avg contacts (%)	**100%**	**39%**	**62.5%**	**37.3%**	**30.6%**	**28.8%**	**15.9%**	**14.2%**

Chapter 3

Numerical Simulation

In this chapter, we describe our simulation app. The same app that we use to simulate different offloading scenarios in the MDC. In this app, we use some real mobile profiles and propose some task capabilities. In fact, those tasks are based on real applications. Moreover, we design several offloading algorithms. Some of them employ for one social factor or a combination of social factors. However, we propose an offloading algorithm to be as lower bound for the offloading process. This algorithm helps us to test for the replication. The term replication here means how many copies of the same task I need to overcome the packet loss or network failure. Furthermore, our focus in the simulation app is measuring 2 basic parameters: completion time (delay) and energy consumption. The completion time is the time needed to complete all created tasks, while energy consumption is the amount of energy needed to execute those tasks. On the other hand, our simulation app is simple and it cannot consider all network conditions. We leave those considerations to be discussed in the next chapter where we introduce our work under a real simulator.

The main goal of our simulation app is to answer the following questions: (i) which type of tasks is worth to be offloaded?, (ii) how to choose an offloadee from a potential list?, (iii) which is the best strategy to follow for offloading a task? and (iv) how many replicas are needed to overcome the task loss?

3.1 Experimental work

As mentioned earlier, our work here is aimed at introducing different offloading scenarios in the MDC. Moreover, we try (numerically) to compare among those scenarios in terms of delay and energy regarding executing some specific tasks. However, one of those scenarios is an algorithm based on our investigation study as described in the previous chapter. In this section, we describe our simulation app in details. Furthermore, we describe the simulation environment by: (i) introducing of basic modules used in the simulation app, (ii) presenting for different mobile profiles and various task capabilities used in our simulation, (iii) defining of various offloading scenarios and offloading algorithms in our app, and (iv) describing of the methodology and metrics used in the simulation.

3.1.1 Platform Description

The idea behind developing this platform (simulation app) is to study the behavior of mobile devices in a MDC. In reality, we cannot perform experiments on very large number of real mobile devices due to monetary and organizational reasons. So, we rely on data provided by several datasets which they based on real experiments. The same tracefiles include basic information related to: (i) number of devices, (ii) interaction among those devices (proximity) and (iii) social factors (if any) between a pair of communicating devices. Here, we use those rich details to propose something close to reality. In our app, we use real mobile device profiles, propose some kind of tasks with some capabilities and consider the interaction and social factors provided by the tracefiles. This helps us in building an interactive environment to test for the offloading process.

Our simulation is a JAVA application consist of the following basic classes:

- *Host*: a class represents for a mobile device. Each host has a profile based on

a specific power (in Joul) and processing complexity or speed (in MFLOP[1]) as described in Table 3.1. This profile is refer to specific device type as illustrated in the same table. Moreover, a host can be an offloader that create some tasks or an offloadee that just execute for a task.

Table 3.1: Device Profiles specifications [30]

Device profile	Total power	Processor	CPU Speed (GFLOPS)	Idle energy per minute (J)
S3	2100 mWh	Quad-core 1.4 GHz Cortex-A9	10.64	7.56
S4	2600 mWh	Quad-core 1.6 GHz Cortex-A15	21.32	6.24
S5	2800 mWh	Quad-core 2.5 GHzKrait 400	160	4.2

- *Task*: a job required to be executed on a Host. Here, the task can be a program unit (i.e. module or function) that can be executed on the offloader or offloadee devices. Furthermore, each task can be defined based on 2 dimensions: size (in MB) and required amount of computation needed to execute such task (in MFLOP). Actually, a task can follow some categories based on the computation amount required to execute that task. Table 3.2 list some of those categories. Moreover, a task require some amount of energy (in Joul) to be executed as we will discussed later. On the other hand, the process of splitting an application into several tasks is out of the scope of this thesis. We leave this as a future direction.

- *OffloadingAlg*: this class performs as a driver class for our app. The same class is used to test all proposed offloading scenarios and algorithms. In the *main* method, we allow the user to enter the number of tasks (N) required to be offloaded. Furthermore, the following is a list of the most important methods in this class:

[1]MFLOP: Milion FLOating-Point, a common measure for the computation speed used to perform floating-point calculations.

Table 3.2: Mapping Tasks to MFLOP[31] based on The linpack benchmark application [32]

Application	MFLOP	Task type
Chess Game (10 move)	10	Low computation
Video Game	30	Medium computation
Object Recognition in Video Feed	60	High computation

- *assignRandomProfile*: a method to randomly assigns profiles to hosts in the designated tracefile. Actually, the profile is selected based on Table 3.1, which includes 3 types of profiles: S3, S4 and S5. Those profiles are based on their corresponding Samsung Galaxy mobile devices. According to the host profile, a host has an amount of computation and energy which is employed to execute task. In other words, an offloadee can execute a task if its remaining energy is above a lower limit which allows the offloadee to survive.
- *generateRandomTaskCapabilities*: this method allocates randomly different tasks capabilities for all N tasks. The capabilities here refer to the size of the task and the amount of computation required to execute the same task.
- *generateRandomTimestamp*: another method to randomly assigned timestamp to each created task. A timestamp is selected within the whole tracefile life (i.e. from the first uptime in the tracefile until the last uptime). However, all created tasks are stored in a list and sorted according to their timestamp.
- *assignRandomTasks*: this method randomly assigns the created tasks to all hosts. So, each host has some tasks waiting to be offloaded to their potential destinations.
- *generateUpDownTimes*: a method to read from a tracefile and fill in the lists of uptimes and downtimes between communicating pairs. Here, uptime is the time when the connection between a pair of nodes is established while downtime is the time when the same connection disconnected. Those lists are

created based on the offloading criteria (i.e. random). Moreover, the same lists are used later to generate another list of potential offloadees available at a task's timestamp.

- *offloadTasks*: a method to offload a task from its initiator (offloader) to a potential offloadee according to the offloading criteria. For example, if the offloading criteria is *Random*, *offloadTasks* send a task to the first offloadee seen by the task's offloader. The offloading process of a task is done according to the task's timestamp. While the time is passed, the app picks up for the next task and tries to offload it to the potential offloadee.
- *generateReports*: a method to generate some reports summarizing what happened during the offloading process. Some of those reports list for the capabilities of the generated tasks, while the others provide some information related to the successfully executed tasks. Moreover, the same method prints out the average consumed energy, average completion time and success rate of applying specific offloading criteria.

3.1.2 Methodology and metrics

In this section, we describe the methodology that we follow and metrics used during the simulation process. Moreover, we describe in details the steps of performing the offloading process and constraints for executing tasks.

In the simulation app we propose the following:

- In our simulation app, we use 3 types of device profiles: S3, S4 and S5. The characteristics of those profiles are listed in Table 3.1.
- We run our simulation in two different environments: homogeneous and heterogeneous environments. In the homogeneous environment, we propose that all devices have the same profile of S3. In fact, this is the lowest settings to see

the performance metrics under this environment. However, in the heterogeneous environment we propose that offloadees have profiles higher than offloaders. In other words, offloaders have S3 device profiles while offloadees have either S4 or S5 profiles.

- For all environments, we suppose that number of offloaders is one-third of the total number of hosts. In other words, if a tracefile has a number of hosts, say M, then the number of offloader hosts will be $1/3M$ and the rest will be offloadees.

- Each tracefile has information about all possible connections between a pair of nodes. This connection is identified by a time which represents the establishment of this connection, we refer to this time as *uptime*. Similarly, the *downtime* is when both nodes are disconnected. We compute the downtime by adding the duration value for this connection to the uptime value (i.e. $downtime = uptime + duration$).

- The focus in our simulation is about two types of tasks: high and medium computation tasks as specified in Table 3.2. We have 2 scenarios here, the first one considers all N tasks of high-computation with 60 MFLOP each. The other scenario proposes that all N tasks of medium-computation with 30 MFLOP. In each scenario, we try to test one single size of tasks (i.e. we select a single size from task's sizes list from 1 MB till 20 MB).

- Before starting the offloading process of a task, the app ensures that the potential offloadee is able to execute the task regarding the amount of remaining energy. Therefore, the offloading process is restricted to a minimum allowed percentage (i.e. 20%) of the offloadee's main amount of energy.

- A pair of devices communicating using Bluetooth link with version 4.0. Table 3.3 illustrates all the characteristics of Bluetooth profile used in our app. Here, we keep the same Bluetooth characteristics to be used in all tracefiles.

Table 3.3: Characteristics of Bluetooth profile used in our app [33]

Category	description
Class/ version	Class 2/ Version 4.0
Radio range	~10 m
Data rate	24 Mbit/s
RTT	3 ms

- In order to compare between local task execution and the remotely execution of the same task, we provide approximations for local task's completion time and consumed energy. Table 3.4 lists for those values for medium-computation tasks on S3 device, while Table 3.5 presents for the same values for high-computation tasks.

Table 3.4: Approximation of Completion time (s) and Energy consumption (J) regarding local execution of medium-computation tasks on S3 devices

Task Size	Completion time (s)	Consumed energy (J)
0	0	0
1	1.4	11.56
2	2.71	19.25
5	6.55	30.38
10	12.09	50.65
20	24.06	87.559

Table 3.5: Approximation of Completion time (s) and Energy consumption (J) regarding local execution of high-computation tasks on S3 devices

Task Size	Completion time (s)	Consumed energy (J)
0	0	0
1	2.61	22.395
2	5.13	40.67
5	13.74	67.475
10	25.2	121.35
20	48.71	175.7

- The total amount of energy (e_c) required to execute a task (T_i) is computed as follows:

$$e_c(T_i) = e_s(T_i) + e_r(T_i) + e_x(T_i) + e_{idle} + e_s(Res) + e_r(Res)$$

where;$e_s(T_i)$: is the amount of energy required to send a task from its initiator to the potential offloadee,

$e_r(T_i)$: represents the amount of energy needed to receive a task at its destination,

$e_x(T_i)$: is the amount of energy required to execute a task at the potential offloadee,

e_{idle}: is the energy consumed per minute while waiting for the reply

$e_s(Res)$: is the amount of energy required to send the result back to the offloader,

$e_r(Res)$: is the amount of energy to receive the results back at the offloader, and

Figure 3.1 illustrates for the energy model used in our simulation.

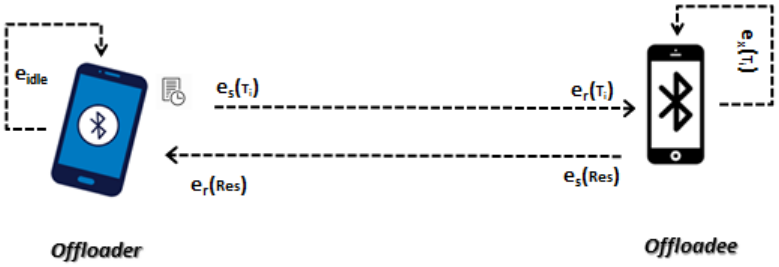

Figure 3.1: Energy model used in our simulation

For fair assumption here, we assume that the energy required to propagate a task and to propagate a result to be zeros. This is because the speed of light imposes a minimum propagation energy on all electromagnetic signals. Moreover, we consider the size of the result to be negligible. As consequences, the amount of energy to send back the result and to receive it on the offloader are zeros. So, the energy model now becomes like this:

$$e_c(T_i) = e_s(T_i) + e_r(T_i) + e_x(T_i) + 0 + 0$$

From the previous formula, we notice that the total amount of energy depends on the three values related to the same task required to be executed. Actually, those values depend on the device's profile. Our contribution here is to expand the work

Table 3.6: Approximation of the time and energy values used in our application for medium-computation tasks

Device	Execution Time [s]	Energy to execute [J]	Idle energy [J]	Energy to send [J]	Energy to receive [J]
S3	24.0	87.6	7.56	15.9	12.8
S4	16.3	50.9	6.24	11.4	10.7
S5	6.0	22.7	4.2	9.8	7.6

Table 3.7: Approximation of the time and energy values used in our application for high-computation tasks

Device	Execution Time [s]	Energy to execute [J]	Idle energy [J]	Energy to send [J]	Energy to receive [J]
S3	48.7	175.7	7.56	27.3	22.9
S4	25.2	79.8	6.24	18.6	14.3
S5	11.7	45.3	4.2	13.9	10.5

as in [34] to include our proposed devices profiles and task capabilities. We provide an approximation of what done in that work. Tables 3.6-3.7 illustrates the amount of energy in all mentioned cases (i.e. to send, receive and compute of a task) and the time required to executed a task on a specific device. Table 3.6 lists for those values for medium-computation tasks while Table 3.7 provides the same values for high-computation tasks.

- The offloading process is done according to the timestamp (t_s) of each task. For parallel execution, we suppose that all tasks start at timestamp t_0, which represents the first timestamp in each tracefile. Therefore, a task will be offloaded from an offloader to a potential offloadee if $t_s \leq uptime(offloader)$; i.e. each task at the offloader will wait for the next uptime in order to start the offloading process. In fact, our simulation app updates the uptime and downtime values after the execution of each execution.

- If the task is successfully executed at the potential offloadee, the completion time interval (t_c) can be computed as follows:

$$t_c(T_i) = t_f(T_i) - t_s(T_i)$$

where:$t_s(T_i)$: task's timestamp (task's starting time). The value of timestamp will be assigned at the beginning of the simulation as described earlier,

$t_f(T_i)$: task's finish time, which represents the time interval after executing the specified task on the potential offloadee and return the results back to the offloader.

More details about the completion time interval can be found here:

$$t_c(T_i) = t_m(T_i) + t_t(T_i) + t_p(T_i) + t_r(T_i) + t_x(T_i) + t_{rm}(T_i) + t_t(Res) + t_p(Res) + t_r(Res)$$

Where:$t_m(T_i)$: is the interval_to_the_first_meet between the offloader and the potential offloadee

$t_p(T_i)$, $t_p(Res)$: are the task's and result's propagation intervals respectively

$t_r(T_i)$, $t_x(T_i)$: the intervals of time to receive and execute the task at the offloadee

$t_{rm}(T_i)$: interval for both communicating hosts to meet again in order to send the results back

$t_t(Res)$: interval of time to transmit the result back to the offloader

$t_r(Res)$: interval of time to receive the result at the offloader

$t_t(T_i)$: task's transfer time, which can be computed as follows:

$$t_t(T_i) = \frac{Size(T_i)}{B}$$

where $Size(T_i)$: is the size of the task in MB as illustrated in tables 3.5-3.7,

B: is the Bluetooth bit rate as described in Table 3.3.

Figure 3.2 illustrates the time interval model used in our simulation.

For fair assumption here, we assume that the time interval required to propagate a task and a to propagate the result to be zeros. This is because the speed of light imposes a minimum propagation time on all electromagnetic signals. Moreover, the size of the result is constant (number) so it can be negligible. As consequences, the interval of time required to send back the result and to receive it on the offloader are zeros. So, the time model now becomes like this:

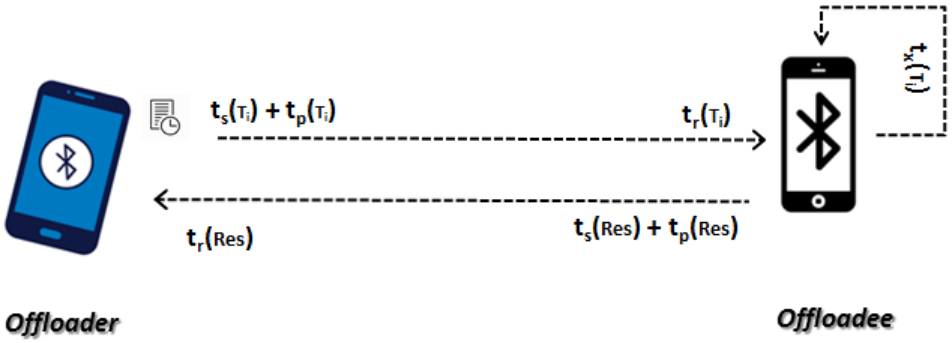

Figure 3.2: Time interval model used in our simulation

$$t_c(T_i) = t_m(T_i) + t_t(T_i) + 0 + t_r(T_i) + t_x(T_i) + t_{rm}(T_i) + 0 + 0 + 0$$

From the previous formula we can notice that the completion time depends mainly on the 3 values. Those values include: task's transfer time from offloader to a potential offloadee and time to receive and execute the same task at the offloadee. Other time intervals depend on the first meet between offloader and offloadee and the re-meeting interval in order to send back the result to the offloader. Actually, those intervals can be high if the uptime between the offloader and offloadee is higher than the timestamp. Moreover, the re-meeting interval can be high if the offloadee node goes away for long time (i.e. offloadee-offloader uptime is high). This is due to the nature of DTN nodes. From here we try to minimize those intervals by designing some social-based algorithms. Nodes share some social aspects are most likely to meet and exchange data.

- In fact, to execute the task T_i at the potential offloadee, we have 3 cases:
  - $t_c(T_i)$ is within the current contact duration (d) between the offloader and the potential offloadee (i.e. $t_c(T_i) \leq d$). This case is ideal for executing some tasks where the computation is not high. Furthermore, it depends on the available connection duration which can be computed as: $d = downtime - uptime$. Here, our app will update the values for uptime and downtime after the execution of each task.

- $t_c(T_i)$ is greater than the current contact duration (i.e. $t_c(T_i) > d$). In this case, the app searches for the next uptime at the offloadee enough to execute the task and ensure the results will return back to offloader.
- Otherwise, T_i cannot be executed within the whole simulation interval. In this case, we mark this task as *unsuccessful* task. Unsuccessful tasks are out of our calculations. Here, we provide an example for this kind of tasks: Suppose we have a task T_3, the same task created at the offloader *H10* with a timestamp equals to (1034) time unit. This task cannot be executed if: (i) there is no available offloadee in the range *H10* after this timestamp until the end of the simulation, or (ii) the duration(s) of the current contact between *H10* and the next offloadee is not enough to execute the task until the end of the simulation.

- Another important metric in our methodology is success rate. Success rate can be defined as the number of tasks that executed successfully at offloadee out of the total number of the whole tasks. This metric can be computed as follows:

$$SuccessRate = \frac{SuccessExecutedTasks\#}{N} \ ;$$

where N: is the number of all tasks assigned to all offloaders.

Our work in the simulation app is toward finding the minimum completion time required to execute each task. In fact, there are different strategies that we can follow to minimize the completion time. Next, we describe some execution scenarios and some algorithms in which social aspects can be employed.

3.1.3 Application Model

Before moving to the offloading process, we describe here our proposed application model in which an offloader has a specific number of tasks waiting to be remotely executed. In our work, we are not proposing any task splitting mechanism, but we try

to work on this in the next future. However, the number of offloader devices form about 33% of the total number of hosts (M). Therefore, each offloader has a number of tasks (u) waiting to be offloaded in parallel to their destinations. In fact, those tasks are considered as independent program modules of an application as illustrated in Figure 3.3. In the same figure, we see that the application has several tasks numbered from 1 till u. Each of those tasks has created at the same timestamp. Here, we have two scenarios for execution: the first scenario considers all tasks to be executed locally at the same offloader device. While the other scenario concerns of offloading the same number of tasks to be executed remotely. As shown in the same figure, *T1* can be offloaded at the same uptime of the offloadee *O1*, while *T2* and *T3* wait for the next uptime of offloadees *O2* and *O3*. In other words, during the simulation: (i) the offloader is busy for executing some local tasks, (ii) when the time is passed, the same offloader has some tasks required to be offloaded in parallel at specific timestamp, (iii) after the offloading process is done, the control returns back to the *main()* module of the same application.

3.1.4 Execution scenarios

An execution scenario here is the way that we follow to execute a task. As mentioned earlier, we consider two types of tasks in our simulation; medium (30 MFLOP) and high (60 MFLOP) computation tasks. This is due to the need to test for the offloading process of those kind of tasks against local execution. To simplify our model, we fix the size of all tasks to be 20 MB. In summary, we have 3 scenarios: (i) local task execution , (ii) offloading of medium-computation tasks, and (iii) local execution of high-computation tasks.

Local task execution For local task execution, we have two scenarios related to the task type (i.e. the computation complexity of those tasks). In the first scenario where

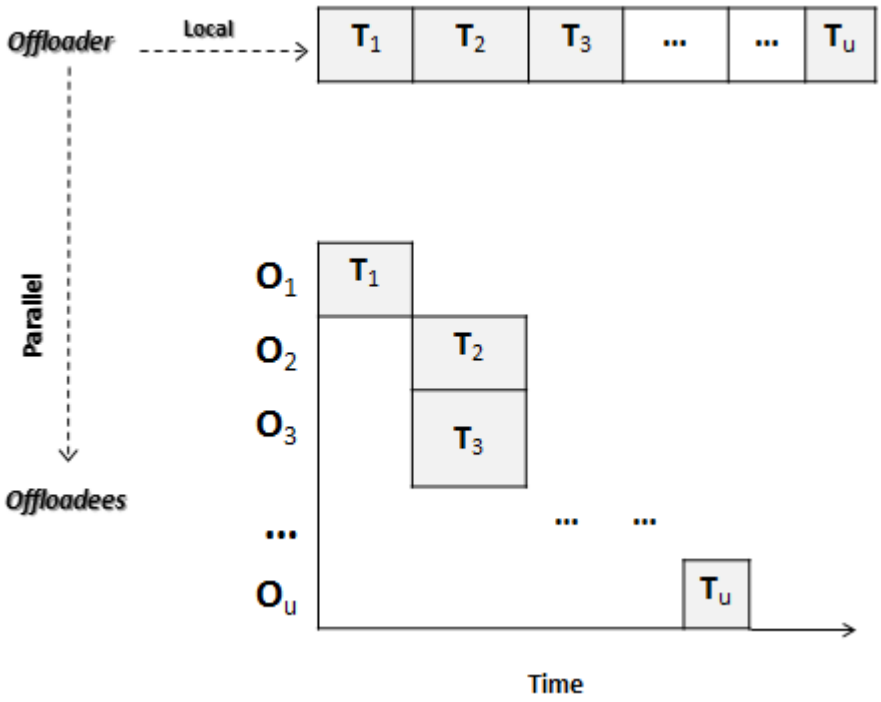

Figure 3.3: Application model for local and parallel execution with u tasks

all tasks are of medium-computation, we propose that an offloader has a number of tasks (u) to be executed locally. As mentioned earlier, we propose all of those tasks are of 20 MB size. The execution of all tasks is done one after the other (i.e. sequential). Then, we measure for the execution time and energy consumption as illustrated in Table 3.4. In fact, we can express about the total execution time and total energy consumption regarding local execution as follows:

$$t_c = u * t_x(T_i)$$

$$e_c = u * e_x(T_i)$$

Where; $t_x(T_i)$: is the time required to execute a task T_i locally according to Table 3.4,

$e_x(T_i)$: is the energy required to execute a task T_i locally according to Table 3.4,

u: is the number of tasks assigned to each offloader. In our simulation we consider the value of u to be from 1 till 6. Here, the value of 1 is the lower limit for the number

of tasks.

For the success rate, we suppose that the offloader device is able to execute all tasks. In fact, we stop until 6 tasks to guarantee this assumption. As consequences, the success rate here is always 100%, or in other words:

$$SuccessRate = 1$$

Similarly, the other local execution scenario where all tasks are of high-computation type is the same. But here we refer to Table 3.5 for the approximations of completion time and energy consumption.

The total number of tasks (N) can be computed as follows:

$$N = u * \frac{M}{3}$$

Where; $\frac{M}{3}$ represents for the number of offloader nodes in the simulation, M here is the total number of hosts in the tracefile.

Medium-computation offloading Here, we propose that all tasks are of medium-computation type. As illustrated in the application model in Figure 3.3, each offloader has a number of tasks u. Those tasks are of the same size with 20 MB and have the same timestamp t_0. All tasks send to be executed in parallel against a number of offloadees. We have the same assumption regarding offloader device profiles which they are of type S3. However, we propose two cases regarding the offloadee device profiles. In the first case we suppose that all devices are of the same profile like offloadees, we call this homogeneous case. The other case called heterogeneous case where the offloadee device profiles are either S4 or S5. In the latter case, we suppose that the offloadee's profile is higher than offloader's profile. This is to see the effect of higher device profiles on the execution performance. The performance metrics are the same as described in Section 3.1.2. In fact, we expect that the completion time (delay) and energy consumption are lower in the heterogeneous case rather than the homogeneous case. This is due to the

time and energy consumption regarding the heterogeneous case is lower (i.e. $t_x(T_i)$ and $e_x(T_i)$ are lower). The success rate is computed as described earlier.

High-computation offloading This scenario is identical to the previous one but here all tasks are of high-computation type. Similarly, the performance metrics are computed as illustrated in Section 3.1.2.

3.1.5 Offloading Algorithms

In order to offload a task from an offloader to a potential offloadee, we design different offloading algorithms. The main goal of this step is to measure numerically the effects of social aspects in the offloading process. Therefore, some of those algorithms include one social factor (i.e. either friendship or interests) while there is another algorithm just offload to a random offloadee. In addition, we design and develop another algorithm that can be considered as lower-bound for the offloading process. We call this algorithm *Flooding*. Particularly, we design another algorithm based on our investigation study described in Chapter 2. In addition, we consider the replication of a task in another algorithm. In the replication algorithm, we try to find a number of replicas enough to reach the performance of the offloading lower-bound.

The main goal of our work here is to provide a clear image about different kind of offloading strategies in terms of time and energy. We test for all those algorithms against several offloading scenarios described earlier. Then, we compare the results against local execution. We can see in figures 3.4 and 3.5 how the offloading decision algorithm works for all offloading strategies. Figure 3.4 illustrates for the definitions used within the algorithm. The same figure include variables definition, messages exchange definitions. Moreover, it includes basic definition of functions used in the same algorithm. Furthermore, Figure 3.5 shows the pseudo-code for our offloading decision algorithm regarding different offloading strategies.

Variables

- *HostProfiles[]: set of host profiles as described in Table 3.1*
- *TaskCapabilities[]: set of task capabilities as illustrated in tables 3.5-3.7*
- *Hosts[]: set of host objects*
- *Tasks[]: set of task objects*
- *s: a number represents the offloading strategy (algorithm). 0: random, 1: Flooding, 2: F-based, 3: I-based, 4: S-based, 5: S-basedRep*
- *u: number of tasks assigned for each offloader. u=1 means sequential execution*
- $e_{main}(H)$*: the main amount of energy at a host H based on Table 3.1.*
- $e_{rem}(H)$*: the remaining amount of energy at a host (after executing some tasks)*
- *P: the minimum percentage energy at the offloadee device allowed for the offloading process (i.e P=20%)*
- *r: number of replicas*
- *simEndTime: last timestamp in the dataset*

Functions

- ***assignRandomProfiles****(Hosts[], HostProfiles[]): randomly assign a profile to every host, the profile is selected from the profiles list*
- ***assignRandomTaskCapabilities****(Tasks[], TaskCapabilities[]): randomly assign some capabilities to all tasks, task capabilities refer to specific sizes and computation*
- ***generateTaskTimestamp****(Tasks[]): generate a timestamp for all tasks, the timestamp is selected randomly based on the dataset duration*
- ***generateUpDownTimes****(s): extract uptimes and downtimes between a pair of hosts based on a strategy s.*
- ***offloadTask****(Task, s, r): offload the task according to the strategy s, list of available offloadees v is generated based on the task's timestamp. If r=0, no replication*
- ***executeTask****(Task, v[i]) :execute a task against an offloadee v[i], size(v[i])>0*
- ***getCurrentOffloadees****(Task): get a list of current offloadees available at task's timestamp*
- ***checkHostProfile****(Host): check if host's profile reach the minimum allowed limit for offloading (here we just deal with remaining host's energy)*
- ***updateHostProfile****(Host): update the host's profile (mainly remaining energy)*
- ***generatingTasksReport****(): generate reports summarizing what happened during the offloading process*
- ***uptime****(Host): returns the current uptime of the host*
- ***downtime****(Host): gets the current downtime of the host*
- ***getNextUptime****(ts): gets the next uptime at the offloadee greater thanor equal to task's timestamp ts*
- ***getNextDowntime****(ts): gets the next downtime at the offloader greater than ts*
- ***executeLocally****(Task): execute a task locally*
- ***executeParallel****(Task[]): send the tasks to be executed in parallel*

Figure 3.4: Pseudo-code definitions of the offloading decision schemes

```
Tasks[] ← generateTasks(TaskCapabilities[])
Hosts[] ← generateRandomProfiles(HostProfiles[])
1: for all T_i ϵ Tasks[i] do
2:     if ts_i ≥uptime(Hosts[i]) then
3:         offloadTask(Tasks[i], s, r)


function offloadTask(Tasks[i], s, u, r)
generateUpDownTimes(s)
offloadees[] ←getCurrentOffloadees(T_i)
1: if checkHostProfile(offloadee) then
2:     if size(offloadees)>0 then
3:         executeTask(T[i], offloadees[j])
4:         updateHostProfile(offloadees[j])


function executeTask(T, H)
t_complete ← t_transfer(T) + t_compute(T)
1: while uptime(H) < simEndTime do
2:     t_up ← getNextUptime(H)
3:     t_down ← getNextDowntime(H)
4:    d ← t_up − t_down
5:     if t_complete ≤ d then
6:         updateUptime(H)
7:         updateDowntime(H)


function checkHostProfile(H)
1: if e_main(H) / e_rem(H) ≥ P then
2:     return true
3: else
4:     return false
```

Figure 3.5: Pseudo-code of the offloading decision schemes

Next, we try to give a brief description of each offloading strategy as follows:

- *Flooding* algorithm, This algorithm tries to offload each task to all available offloadees at the timestamp t_s of a task T_i. In this algorithm, the same copy of the task T_i is sending to all available offloadees at t_s and recording for the completion time $t_c$and energy e_c. Then, we consider the execution of a copy with minimum t_c. We did this for all N tasks and then we compute for the success rate. Actually, we consider this algorithm as lower-bound for the offloading process.

- *Random* offloading, offload the task T_i to a randomly selected offloadee available at timestamp t_s. In fact, the selection process is done randomly from the list of available offloadees at t_s. Here, we just offload a task to the first encountered offloadee whatever the social relation exist or no.

- *Friendship (F-based)* offloading: here the offloading decision is done by considering the friendship factor. As described earlier, friendship here means either both pair of hosts are friends (direct friendship) and they share some common friends. Actually, this algorithm offload to the first encountered offloadee that satisfies the friendship condition.

- *Interests (I-based)* offloading: here we just consider for a number of common interests between an offloader and a potential offloadee. Moreover, the offloading process is done to the first offloadee satisfies the interests constraint.

- *Investigation Study (S-based)* offloading: the offloading decision here is done by considering the outcome of our investigation study as described at the end of Chapter 2. The selection of an offloadee is done under 2 conditions; (i) if both pair of hosts are friends and share at least 4 common interests, or (ii) both hosts just share at least 4 common interests.

- *Replication (S-basedRep)* offloading: here we test for the replication by considering

our investigation study. We started the replication from 2 until 6 to see which number replicas close to the lower-bound.

However, we design for all set of algorithms mentioned earlier. We redefine for the friendship offloading to include either direct friendship between a pair of hosts or common friends between the same pair. The algorithm which is based on common interests is already exist. Moreover, random offloading is already exist [34]. Other set of algorithms we have completely designed.

3.2 Experimental Results

We provide results regarding two cases. In the first one we propose a homogeneous case where all device profiles are of type S3 and all tasks are of type medium computation. Here, we fix the task size to be 20 MB and we set the number of tasks at offloader (u) to be from 1 till 6. In the second case, we just change the device profile for offloadees to include S4 and S5 profiles (heterogeneous case). For both cases, we keep the offloaders ratio as the same with 1/3 of the total number of devices. Furthermore, we try to apply both cases on high-computation tasks. However, the completion time and energy consumption of the whole system are computed based on the same models provided earlier in Section 3.1.2. Here, we consider the size of the result for executing a task to be zero. As consequences, the time required to send the result back to the offloader and the time required to receive the results at the offloader will be considered as zero. Similarly, the energy consumption for both purposes are considered as zeroes. In addition, the propagation time will be considered as zero.

We present the results here regarding Sigcomm09AT dataset. This dataset includes 76 hosts communicating together for 3 continuous days. Moreover, we illustrate the results regarding all designed algorithms. Then, we compare between local tasks execution and parallel execution in terms of time and energy. Another important measure among offloading strategies is the success rate. An algorithm with highest success rate can be considered as more stable criteria for offloading process.

3.2.1 Case of homogeneous environment

In the homogeneous case, we deal with two type of tasks: medium-computation and high-computation task.

Medium-computation tasks scenario First, we present the results regarding medium-computation tasks as shown in figures 3.6-3.8. Figure 3.6 illustrates the success rate

as percentage respect to the total number of tasks. In the same figure, we see how the success rate values decreased by increasing the number of tasks to be executed in parallel. This behavior refers to the fact that a pair of nodes in a DTN network can exchange information at a specific amount of time, then they move away for an amount of time. So, the offloadee in this case may not encounter the offloader in order to send the result back. According to that, some tasks are marked as un-executed tasks and this will decrease the success rate value. Furthermore, in the same figure we can see which criteria is the best regarding the success rate values. We can see here how the success rate for the *Flooding* and *S-basedRep* are the highest with success rate until 80% for $u = 6$. The second best is *S-based* which is close to *S-basedRep*. Then, we can see how *I-based* and *F-based* come after. Finally, *Random* gives the lowest success rate until 50% for $u = 6$. The main conclusion from this plot is: social aspects between a pair of nodes play a crucial role in increasing the possibility to execute a task. Nodes share some social aspects are more likely to meet and exchange information. Moreover, *S-basedRep* and *S-based* are considered as best protocols to achieve this.

Figure 3.7 plots the completion time (delay) for parallel execution of a number of tasks (u). We can see from the same figure how the tasks completion time increased dramatically for local execution. This is due to the fact that in local execution tasks will be executed in sequential as illustrated in Figure 3.3. In other words, T_1, T_2 until T_u are executed one after another. The total delay for executing u tasks can be computed according to Section 3.1.2. We can see from the same figure that the delay is also increased regarding parallel offloading by considering different offloading protocols. At lower limit of tasks ($u = 1$), we can see how the delay regarding local execution is the lowest. This is due to the fact that in the homogeneous case the delay can be computed as the time required to execute that task. While in offloading protocols there is also amount of time required to send and receive the task form the offloader to the potential offloadee. Regarding parallel execution, we notice that the *Flooding* protocol

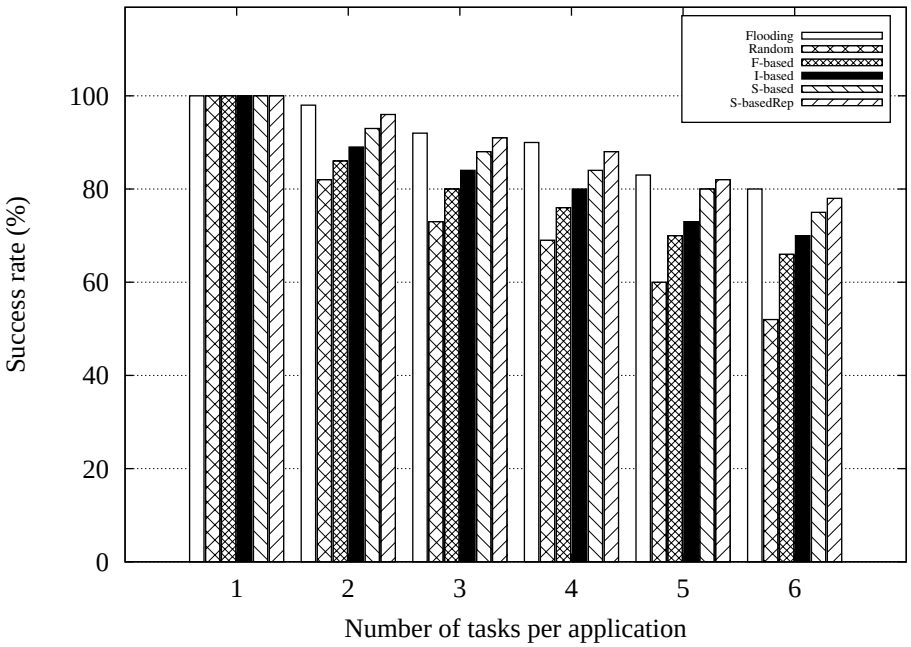

Figure 3.6: Success rate regarding parallel execution of u medium-computation tasks (Homogeneous Case)

is the best is conserving time. The same protocol gives the lowest values for delay regarding number of tasks u. *Flooding* protocol saves up to 65% in time regarding local execution. The second best is *S-basedRep* which saves more than 55% regarding local execution. Then, the advantage goes to *S-based* protocol with around 45% time saving respect to local execution. After that, *I-based* and *F-based* saves around 35% and 30% respectively. Finally, Random protocol gives the highest delay but it can save up to 15% in time regarding local execution.

Moving to the energy consumption as shown in Figure 3.8, we can see how the energy consumption of local execution of u tasks goes in linear way. In fact, for local task execution we just consider the energy required to execute the same task on the offloader. Moreover, we notice that in the lower limit where $u = 1$that the local execution have the advantage regarding offloading. This is due to the fact that there is amount of energy required to send or receive the task in different offloading scenarios. The total energy consumption is computed as described in Section 3.1.2. Furthermore, we can see

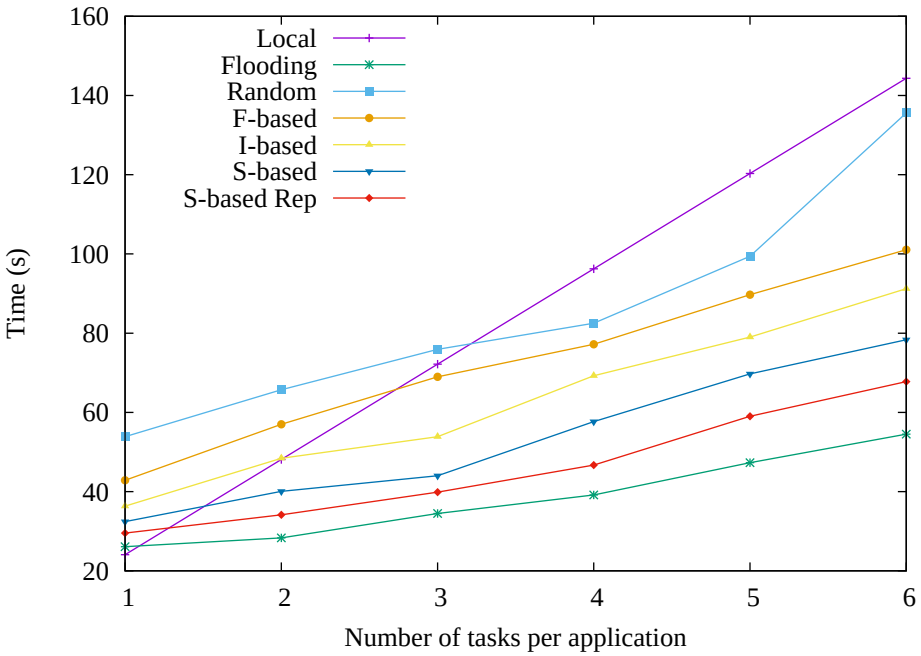

Figure 3.7: Completion time regarding parallel execution of u medium-computation tasks (Homogeneous Case)

from the same figure how the energy consumption for *Random* protocol is the lowest. This is due to the fact that in *Random* protocol the offloader send the task to the first encountered offloadee whatever that offloadee is. In other words, the offloadee is close to the offloadee other than offloading protocols. *Random* protocol saves up to 50% regarding local execution. Then, the energy saving goes to *I-based* and *F-based* protocols with around 40% and 35% respectively. After that, *S-based* protocol comes with about 25% saving in energy. Finally, *S-basedRep* and *Flooding* are the worst in energy saving. This is due to the fact that the amount of energy to send and receive the same copy of the task is multiple. However, both protocols still save some amount of energy with around 15% and 10% respectively.

High-computation tasks scenario High-computation scenario results are illustrated in figures 3.9-3.11. In Figure 3.9, we can see how the success rate decreased compared to Figure 3.6. This is due to the fact that the task computation here is

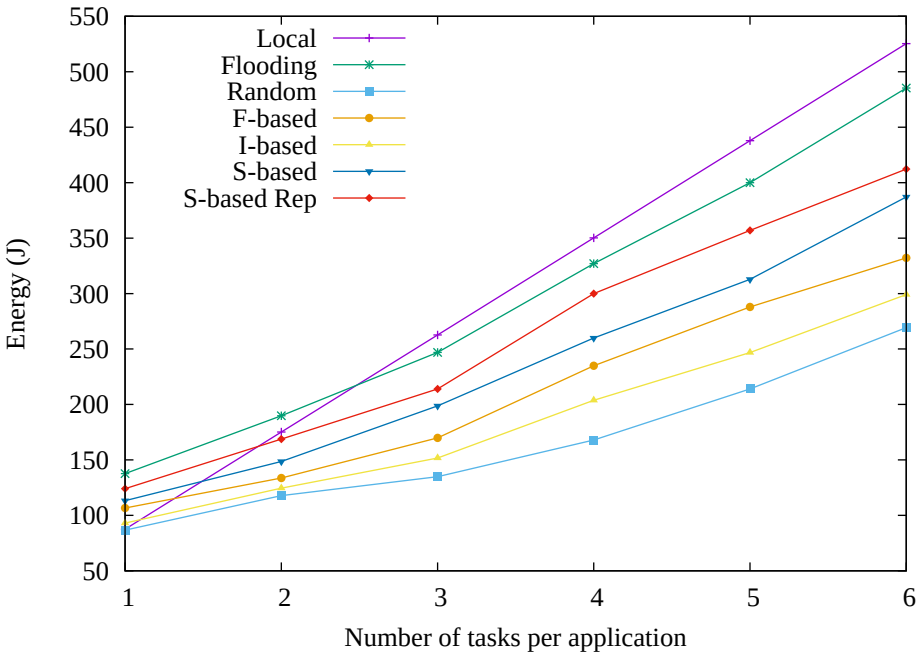

Figure 3.8: Energy Consumption regarding parallel execution of u medium-computation tasks (Homogeneous Case)

higher so it's less likely to be executed against different offloading criteria. From the same figure, we can see how *Flooding* and *S-basedRep* are in the top with higher success rates respect to the other offloading protocols. Same protocols still have around 65% as success rate each for $u = 6$. S-based follow them with a success rate of more than 60%. Then, *I-based* and *F-based* comes with success rate exceeds 55% each. Finally, Random protocol comes with a success rate of less than 50%.

Delay is shown in Figure 3.10 which indicates almost the same behavior as in Figure 3.7. Here, the delay is higher due to the execution time required to execute this type of tasks is higher. We can see from the same figure how the local execution goes in a linear way respect to the number of tasks (u). We notice that at lower limit (i.e. $u = 1$) how the local execution gives the lowest delay, this is because there is no amount of time to send or receive the task like in different offloading scenarios. The same figure indicates that *Flooding* is the best strategy with more than 55% saving in time regarding local execution, followed by *S-basedRep* with around 50% saving in time compared to local

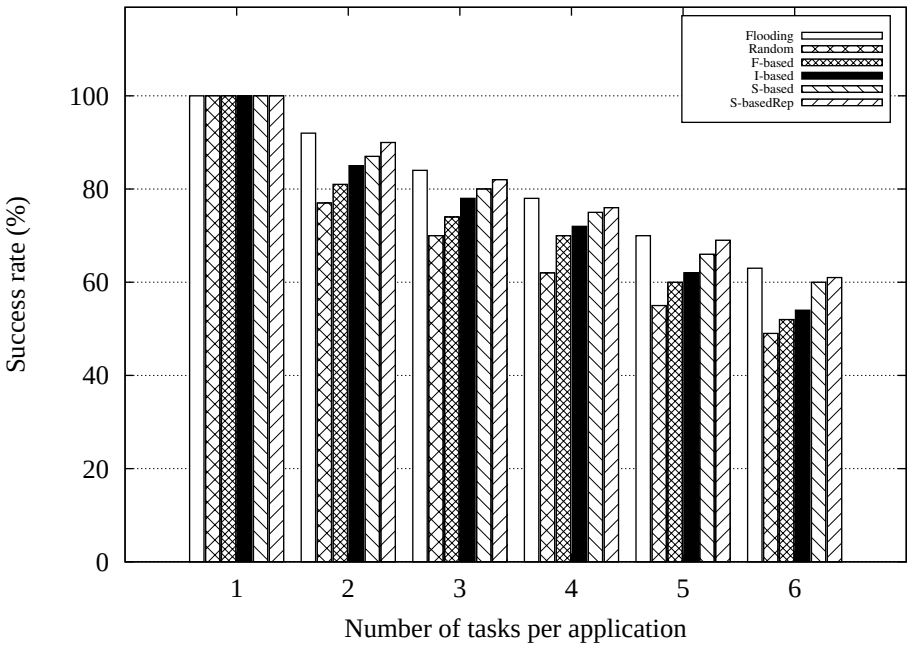

Figure 3.9: Success rate regarding parallel execution of u high-computation tasks (Homogeneous Case)

execution. Then, S-based comes with around 40%. *I-based* and *F-based* come after with around 30% and 25% respectively. At the end, *Random* gives the lowest saving with 10% compared to local execution.

Moving to energy consumption as shown in Figure 3.11, we can see higher energy values here compared to the same as in Figure 3.8. Local execution maintains the same linear behavior here, where the local energy consumption is dramatically increased by increasing the number of tasks (u). Moreover, we can notice that at $u = 1$ local execution is still gives the lowest energy compared to offloading. In the same figure, we can see how Random protocol gives the lowest values for energy consumption with a saving of more than 75% regarding local execution. *I-based* and *F-based* protocols comes later with energy savings of about 70% each. Then, *S-based* comes with about 65% energy saving regarding local execution. After that *S-basedRep* comes with energy saving of about 60% regarding local execution. Finally, *Flooding* protocol gives more than 50% energy saving respect to local execution.

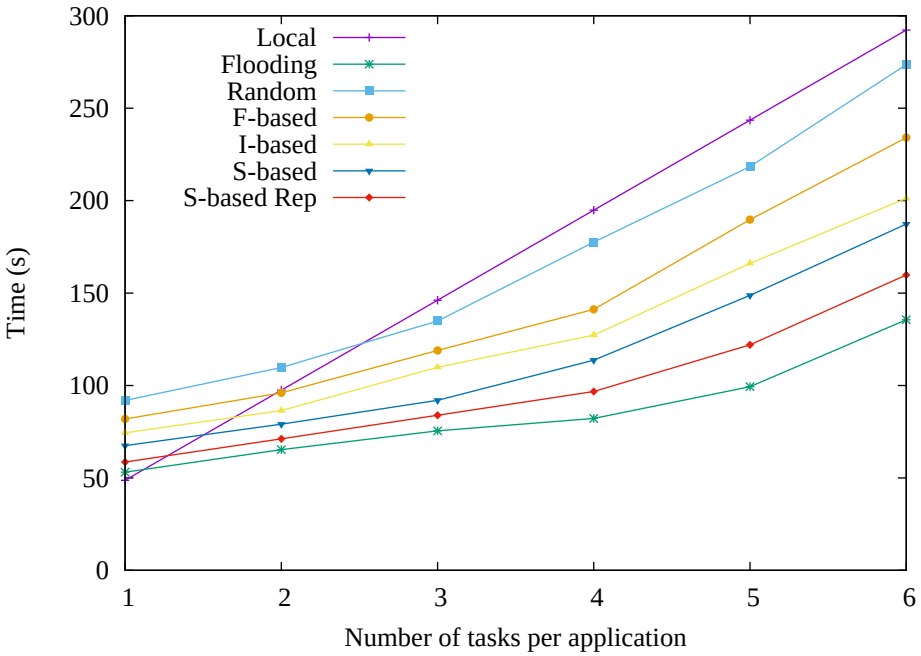

Figure 3.10: Completion time regarding parallel execution of u high-computation tasks (Homogeneous Case)

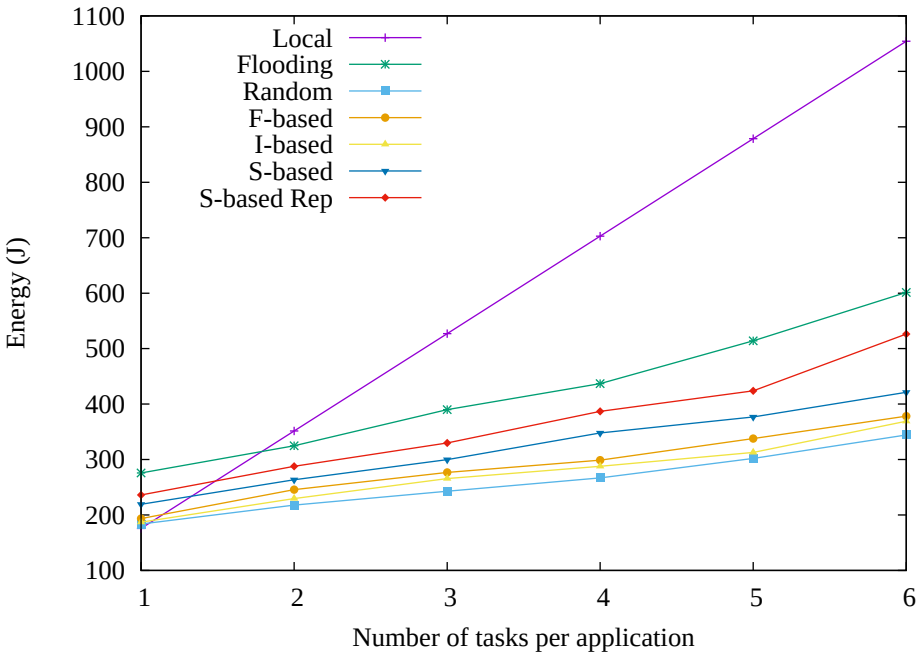

Figure 3.11: Energy Consumption regarding parallel execution of u high-computation tasks (Homogeneous Case)

3.2.2 Case of heterogeneous environment

Similarly, in the homogeneous case we deal with two type of tasks: medium-computation and high-computation task.

Medium-computation tasks scenario Figures 3.12-3.15 illustrate the results of medium-computation offloading regarding the homogeneous environment. We can see from Figure 3.12 how the success rate of *S-basedRep* is the highest regarding other offloading protocols and it is very close to the values for the lower-bound (*Flooding*). For both protocols the success rate for 4 exceeds 90%, which means that almost all assigned tasks are successfully executed remotely. This percentage goes a little bit down for $u > 4$, but it is still above 80%. Moreover, *S-based* protocol is not far away from replication version and gives around 80% for $u = 6$. On the other hand, *Random* protocol gives the lowest success rate with around 50% for $u = 6$. Furthermore, *I-based* protocol gives higher success rate compared to *F-based*,but both of them are still below *S-based*.

Figure 3.13 plots for the completion time regarding all set of protocols. In the same figure, we can see how the time for local execution increased dramatically respect to the number of assigned tasks. From the same figure, we show how the delivery time regarding the *Flooding* and *S-basedRep* protocols give the lowest values. *S-basedRep* protocol saves more than 75% of delivery time regarding local execution. In fact, *Flooding* and *S-basedRep* protocols try to send several copies of the same task and wait for the one with shorter execution time. Moreover, *S-based* protocol saves around 60% respect to local execution. However, other offloading criteria save some amount of time but after number of tasks greater than 2. We show this as in *I-based* and *F-based* protocols, which they save around 35% and 30% respectively. Finally, *Random* protocol saves some amount of time with about 15% for number of tasks greater than 3.

In Figure 3.14 we can see how the energy consumption regarding all designed pro-

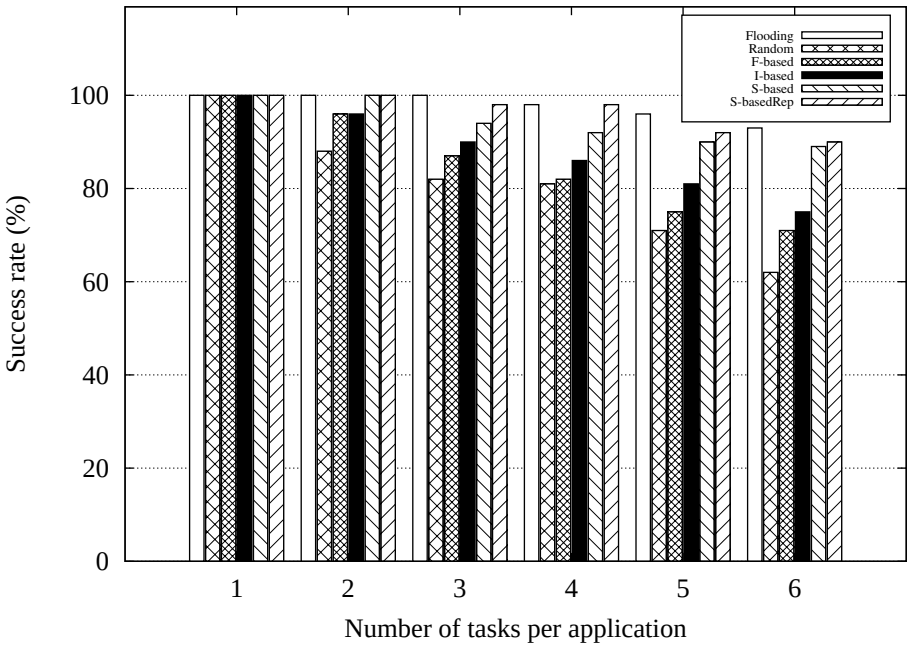

Figure 3.12: Success rate regarding parallel execution of u medium-computation tasks (Heterogeneous Case)

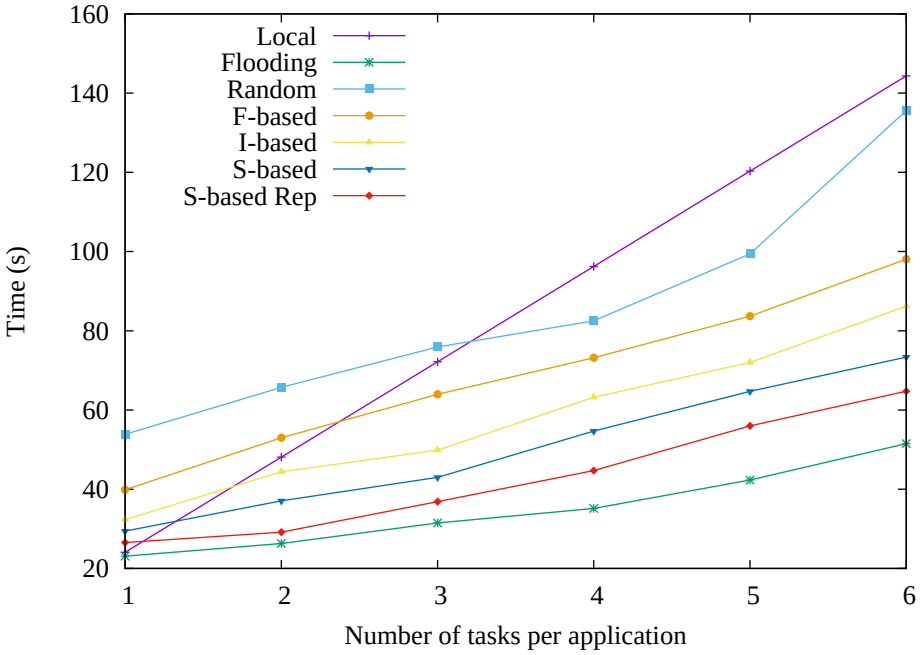

Figure 3.13: Completion time regarding parallel execution of u medium-computation tasks (Heterogeneous Case)

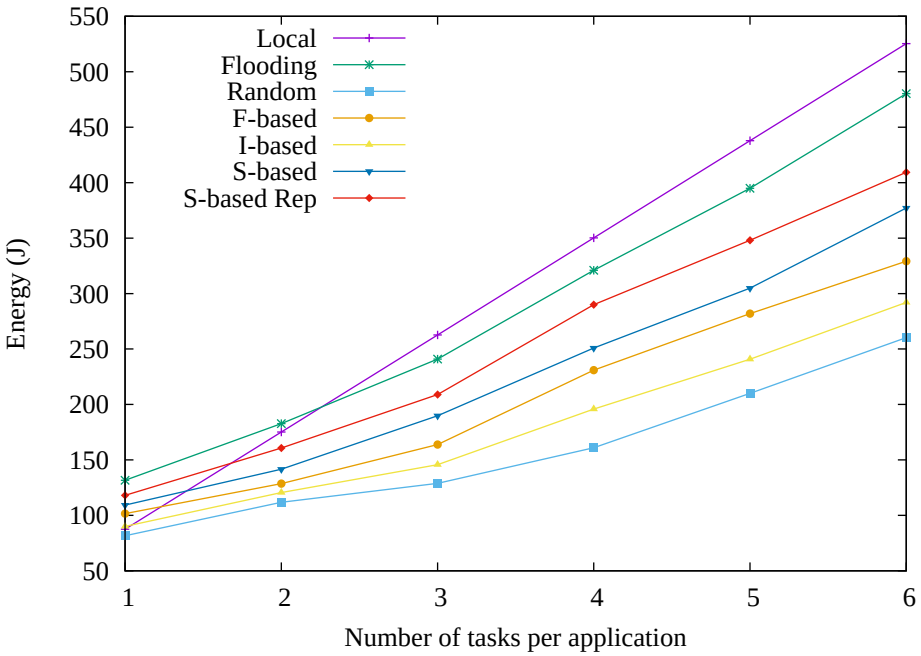

Figure 3.14: Energy Consumption regarding parallel execution of u medium-computation tasks (Heterogeneous Case)

tocols. From the same figure, we see how the energy consumption regarding local execution increased in a linear way. Moreover, we notice that *Random* protocol gives the lowest values for energy with about 60% saving in energy comparing to local execution. Furthermore, *I-based* and *F-based* saves amount of energy equals to about 50% and 45% respectively compared to local execution. Then, *S-based* and *S-basedRep* save about 35% and 30% respectively compared to local execution. Finally, *Flooding* protocol is the worst in energy consumption among all other protocols, but the same protocol still saves around 20% regarding local execution. This can be referred to the amount of energy used to transmit the same task to all available offloadees.

High-computation tasks scenario The results of high-computation tasks under heterogeneous environment are illustrated in figures 3.15-3.17. In Figure 3.15, we see how the success rate decreased compared to the same figures for medium-computation offloading. In other words, the number of successfully executed tasks of type high-

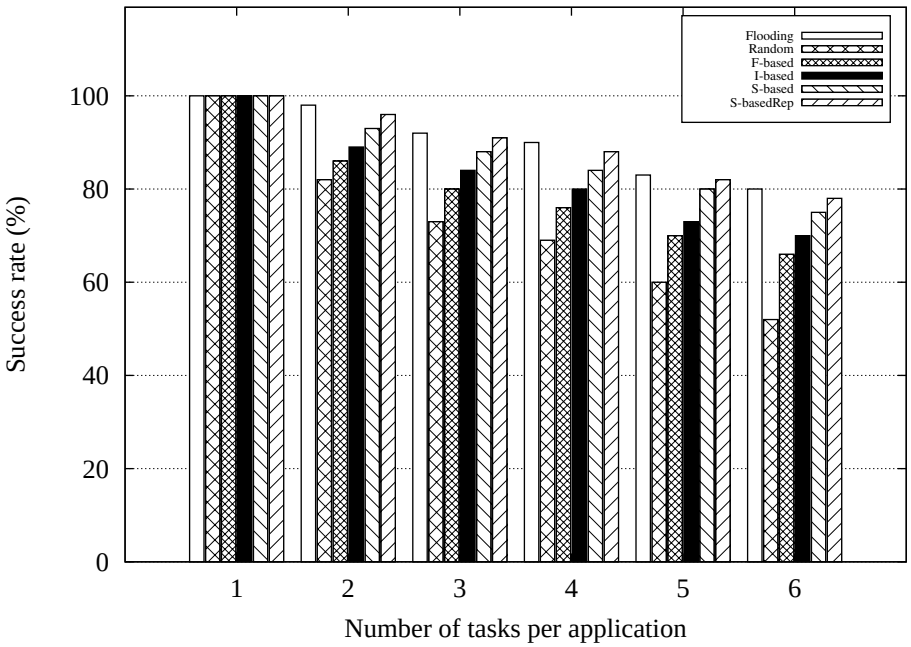

Figure 3.15: Success rate regarding parallel execution of u high-computation tasks (Heterogeneous Case)

computation is lower than of type medium-computation. From the same figure, we notice that *Random* protocol gives the lowest success rate with just 60% for $u = 6$. However, *S-basedRep* is still the best strategy to follow and it is very close to lower-bound. Moreover, *S-based* can be considered the second best strategy here. Other protocols are behave the same, *I-based* gives higher success rate compared to *F-based*.

Respect to delivery time as shown in Figure 3.16, we still have the same view here. *S-basedRep* gives the lowest delivery time regarding local execution. The same protocol is very close to the lower-bound and it saves more than 55% of time. Furthermore, S-based saves around 45% of time respect to local execution. I-based and F-based save about 30% and 25% respectively. Moreover, *Random* offloading saves some of amount time with about 15%.

In Figure 3.17, we can see how *Random* offloading saves more than 75% of energy respect to local execution. *I-based*, *F-based* and *S-based* protocols are very close to each others and saves around 60% regarding local execution. Furthermore, *S-basedRep* saves

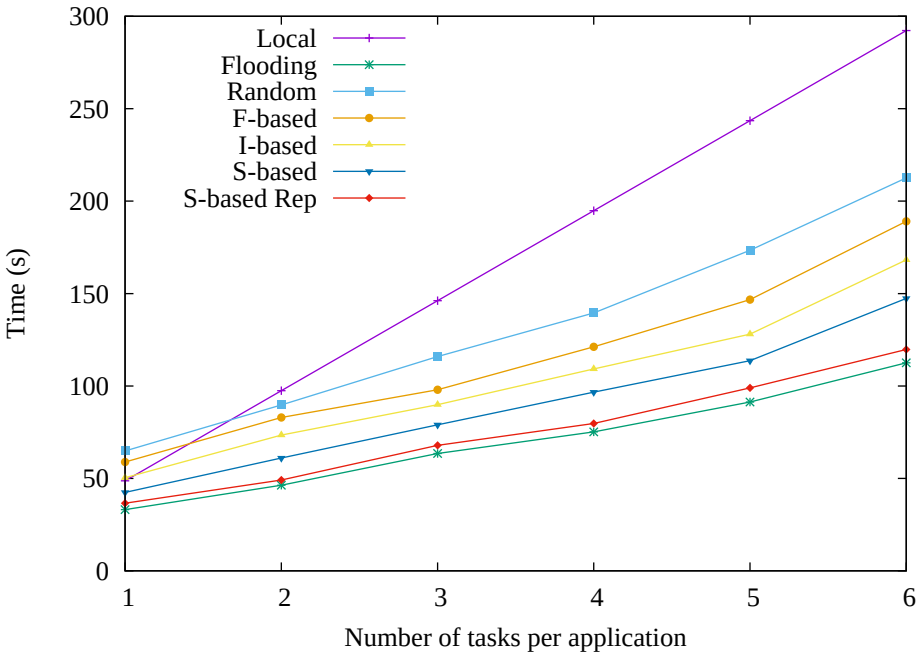

Figure 3.16: Completion time regarding parallel execution of u high-computation tasks (Heterogeneous Case)

around 45% while *Flooding* protocol saves about 30% regarding energy consumption of local execution.

In brief, we build a simple app to test for different offloading algorithms. Moreover, we test for the replication factor against lower-bound offloading. Our outcome from this simulation indicates that *S-based* offloading is the best in conserving execution time. However, *Flooding* provides a lower-bound for the offloading by considering all offloadees available at task's timestamp. Furthermore, *Flooding* is the worst in energy consumption.

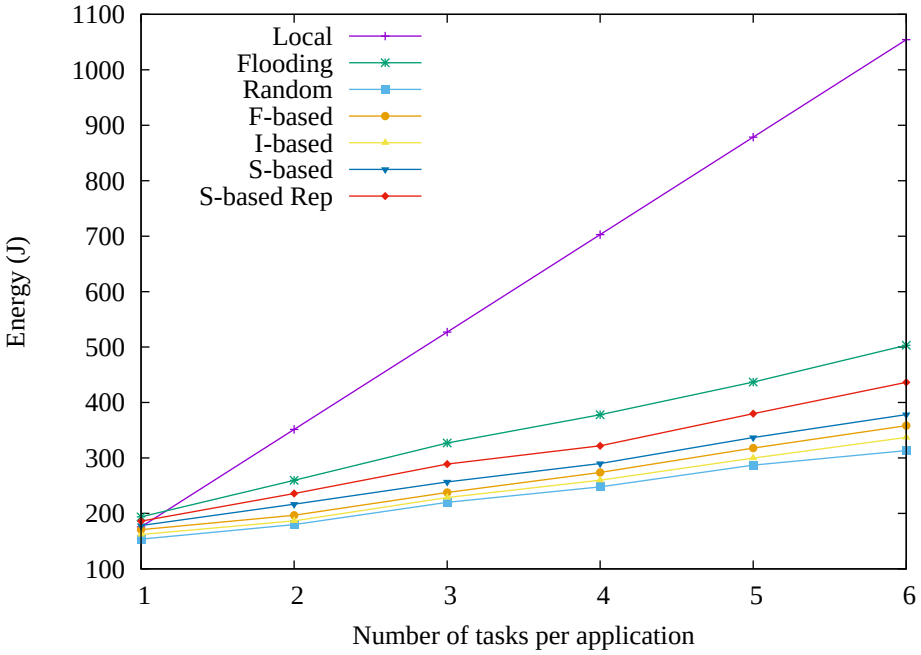

Figure 3.17: Energy Consumption regarding parallel execution of u high-computation tasks (Heterogeneous Case)

Chapter 4

Real Simulation

In this chapter, we try to test our model in real simulation environment. This environment include all network conditions (i.e. packet drop, congestion, etc.). In order to achieve that, we use the ONE Simulator [35] as a simulator used mainly for opportunistic network. Moreover, we try here to test for all proposed scenarios described earlier against different offloading algorithms. In addition, we try to keep using the same host profiles and some task capabilities as described in the previous chapter. Furthermore, we use the report generator module within the same simulator to provide some statistics and results about the offloading process. However, here we use only *Sigcomm09AT* dataset under this simulation. This is due to the continuity characteristics of this dataset (i.e. ther is no gaps within days). Here, we measure the delivery time and energy consumption of u tasks assigned to each offloader within the simulation.

4.1 Simulation environment

In order to test and evaluate our investigation study as described in Chapter 2, we use the ONE Simulator. This simulator is considered as an opportunistic network environment simulator (i.e. is used mainly for DTN network). Furethermore, it can import data from real-world mobility traces and able the user to generate different

kind of reports related to node movements, message passes and general statistics. The simulator has been developed under support of Nokia Research Center, Finland. Here, we list for the basic characteristics and functions of this simulator as follows[36]:

- The use of different built-in movement models and allowing the researcher to build their movement models. This is help to test for node movement scenarios within the same environment.
- Support for various pre-existing DTN routing algorithms for delivering messages between nodes and allowing researcher to build their custom routing algorithms in the simulation environment.
- Providing with a GUI to visualize the mobility and message delivery in real-time with the support for different reporting according to the researcher's need.
- An open source JAVA-based tool; this is useful for creating and implementing custom applications and testing for specific models and scenarios.
- Simple configuration through text file; for example, it allows to configure for the number of nodes in the scenario and Bluetooth communication interface by just fixing those values in configuration file

Figure 4.1 illustrates for the architecture of the ONE simulator. More details regarding this architecture can be found in [36].

Regarding our work under the ONE simulator, we develop 2 types of applications: *OffloaderApp* and *OffloadeeApp*. Those applications simulates for an offloader and offloadee roles respectively. In addition, we update some existing core classes within the ONE Simulator in order to meet our requirements. Here, we mainly update the *Message* class in order to handle our proposed *Task* and *DTNHost* class to deal with *Host*. Moreover, we describe the metrics and methodology used during the simulation. Furthermore, we describe scenarios used to test offloading algorithms proposed earlier.

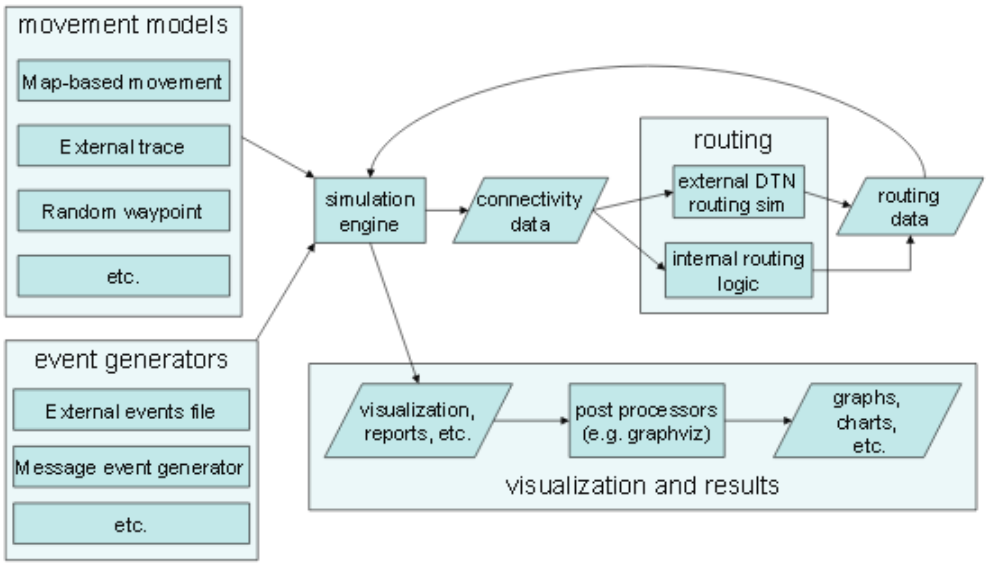

Figure 4.1: The ONE simulator architecture [35]

4.2 Experimental work

The aim of using this kind of simulation is to test our proposed offloading algorithms in real environment under all possible network conditions. In this section, we describe in details our proposed scenarios under this simulator and the our application and modules built within the same simulator.

4.2.1 Scenario configurations

The scenario here refers to the simulation scenario within the same simulator. In fact, the same scenario can be easily configured by setting some parameters in a text file. So, we can set those parameters in the *default_settings.txt* (a default configuration file) provided by the ONE simulator. In this file we can configure different settings, like:

- The duration of the simulation
- Some specific settings related to the interface. For instance, user can set the transmit speed (bytes per second) and transmit range of the interface (meters).

- Bluetooth interface settings, including: Bluetooth interface, transmission speed and transmission range
- Different group of nodes. This is useful in our case to distinguish between offloader nodes and offloadee nodes.
- Message settings, including message creation parameters and event generators.
- Movement model settings, here the user can identify the world's size for movement models and either user of built-in or custom movement models.
- Routing settings for built-in or custom routing modules.
- Reports setting, here user can set for the number of reports and for each report either to state the built-in report module or custom report module. Moreover, the user can indicate the default directory of reports.
- GUI settings, here user can set for different GUI underlay image, including: source file, offset and the scale of such image.

Next, we describe the settings regarding our simulation scenario under the ONE simulator.

4.2.1.1 General configurations

Here, we identify for basic configurations in our scenario used under the ONE Simulator.

World size refers to the proposed region to apply for a scenario in meters. For our scenario, we set the world size to 120, 100. Those values refer to the length and width of the region respectively.

Scenario duration the whole scenario duration in seconds. For our work, we propose that this value is equivalent to *Sigcomm09AT* dataset duration which is 3 days. Table

Table 4.1: Simulation scenario basic parameters

parameter	description	value (s)
name	title of the scenario, appeared in the running window	MDC_Scenario
startTime	simulation start time	1250488830
endTime	simulation end time	1250719161
nrofHostGroups	number of nodes groups to be created (at least one)	3

Table 4.2: Bluetooth settings used in our scenario

parameter	description	value
type	Bluetooth interface for all nodes	SimpleBroadcastInterface
transmitSpeed	Transmit speed of	3072k
transmitRange	Transmit range of	10

4.1 summarizes for the basic settings of our proposed scenario. From the same table, we see how we can indicate a name for our scenario (i.e. MDC_Scenario). This name is accompanied by any file generate during the same simulation (i.e. reports). The simulation scenario is identified by two timestamps representing the start and end of the simulation. Moreover, we can define how many group of nodes in our simulation. Here, we use 3 type of groups, one for offloader group and the others for offloadee groups.

4.2.1.2 Bluetooth configuration

Here, we propose the use of Bluetooth 4.0 with a speed of 24 Mbps and a range of 10 meters as illustrated in Table 4.2. The Bluetooth interface here is used to simulate the Bluetooth communication between nodes.

4.2.1.3 Group settings

In our scenario, we have three groups of nodes; the first group represents offloader group. This group of nodes use *OffloaderApp* module (a custom application built under the ONE simulator to deal with offloader nodes). Furthermore, the last two groups represent offloadees groups. This is because we have two offloadee profiles: S4

Table 4.3: Basic settings for all group of nodes used in our scenario

parameter	description	Group1	Group2	Group3
nrofHosts	No. of nodes in the group	25	25	26
groupID	Identifier for the node in the current group	R	E	E
router	Routing protocol	Random, Ibased, Fbased, Sbased, Sbasedrep	Random, Ibased, Fbased, Sbased, Sbasedrep	Random, Ibased, Fbased, Sbased, Sbasedrep
nrofApplications	No. of applications to be set on the group	1	1	1
application1	Name of applications to be set on the group	R_App	E_App	E_App
TaskApp	Taskapplication running at the beginning of the scenario	-	-	-

and S5. Both of those groups use *offloadeeApp* module (another custom application to deal with offloadee nodes). We set the number of offloader nodes to be 25 and number of offloadees to be 51. The total of 76 which reflects the total number of nodes in *Sigcomm09AT* dataset. As illustrated in Table 4.3, we try to make a fair distribution of proposed profiles against total number of nodes. The most important thing here is we can simulate our proposed protocols as routing protocols to deliver messages to their destinations. Here, we test for all set of those protocols to measure the performance parameters described earlier.

Table 4.4 list for the basic settings of nodes used in our simulation. Here, we didn't have any information about the nodes location, so we keep them as default. In other words, we let the simulator to randomly spread node within the simulation area. Those nodes are moving randomly using an existing movement model called RandomWaypoint. In the future we aim at building a custom movement model to consider each selection criteria. Moreover, we suppose that all nodes have the same Bluetooth interface as

Table 4.4: Common settings of all nodes in our scenario

parameter	description	value
movementModel	Mobility model for all of the nodes	RandomWaypoint
nodeLocation	Group node location	0,1
router	Routing protocol to be used by each node in the group	Random, Ibased, Fbased, Sbased, Sbasedrep
bufferSize	Buffer size of each node	100M
interface1	All nodes have the bluetooth interface	btInterface

Table 4.5: Message creation parameters

parameter	description	value
Events.nrof	number of event generators	2
Events1.class	Class of the first event generator	ExternalEventsQueue
Events2.class	Class of the second event generator	ExternalEventsQueue
Events1.filePath	Path of the external messages file	traces/Sigcomm09AT.txt
Events2.filePath	Path of the external messages file	traces/Sigcomm09AT.txt
size	message size	20

described in Table 4.2. Furthermore, each node is provided with a 100 MB buffer in order to store messages upon execution.

4.2.1.4 Message settings

Our scenario consists of two event generators, both of them used to describe messages (tasks) used. Here, we consider medium and high computation tasks, so we have 2 events. Actually, the simulator allows us to set the message creation interval. We set the interval for both events to be within the whole scenario period. In other words, we let the ONE simulator to read the information in the *Sigcomm09AT* tracefile by specifying the file path as illustrated in Table 4.5. Furthermore, we set the task size to be 20 MB as we study this size in the numerical simulation.

Table 4.6: Energy values in units at offloader and offloadee nodes [38]

Energy type	description	offloader value (S3)	S4 offloadee value	S5 offloadee value
initialEnergy	initial budget of units at each node	1000	1500	1800
scanEnergy	units usage per second in device discovery	29	29	29
scanResponseEnergy	units usage per second in device discovery response	0	0	0
baseEnergy	amount of energy when the node is idle	0.05	0.05	0.05

4.2.1.5 Energy settings

The One Simulator consider energy budget approach to model the host's energy consumption . This amount of energy will be used during each host's activities (i.e. scanning, transmission and receiving messsags). Table 4.6 illustrates the energy values during different activities. In the same table, we consider different budgets for different node profiles.

4.2.1.6 Movement model

Movement model describes the mobility of nodes within their world (space). We consider the *RandomWaypoint* mobility in our simulation scenario. In fact, this movement model is not the best to be used in our scenario. However, as future work we try to build this kind of model to reflect interaction between nodes sharing some social aspects. Nodes sharing some kind of social aspects are close to each other and more likely to meet.

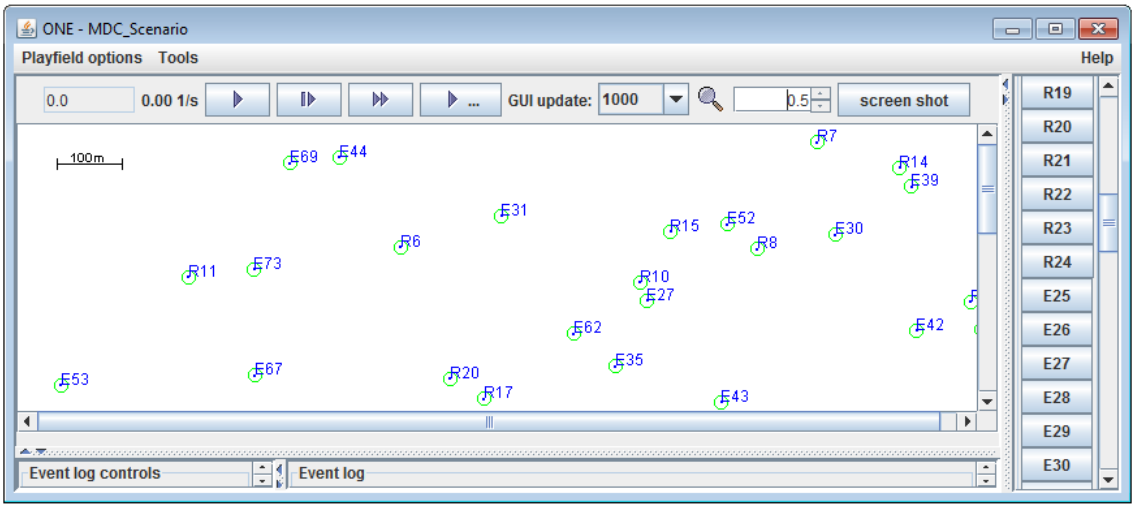

Figure 4.2: Initial scenario of our work

4.2.1.7 Routing settings

Here, our simulation scenario delivers a message (task) based on the offloading algorithm used. In other words, if the offloading algorithm is *Random*, the message delivers to the first encountered host. Otherwise, if the offloading algorithm is *I-based*, the message can be delivered if the encountered host share some interests with the offloader.

Figure 1 illustrates the initial scenario of our work just before running on the ONE simulator. From the same figure, we notice how the simulator initially spread the nodes within their predefined world. Furthermore, we refer to an offloader with a letter *O* followed by a serial number forming its id. The same thing with offloadees but with a letter *E*.

4.2.1.8 Generated reports

we update some existing report modules within the ONE simulator in order to handle our results. Here, we basically update the following report modules: *MessageStatsReport*, *EnergyLevelReport*, *DeliveredMessagesReport*. Actually, *DeliveredMessagesReport* states some information about successfuly executed tasks including execution time for each with the average execution time at the end. While *EnergyLevelReport* reports for

Table 4.7: Report setting in our scenario

parameter	description	value
nrofReports	how many reports to load	3
reportDir	default directory of reports	reports/MDC
report1	first report	MessageStatsReport
report2	second report	EnergyLevelReport
report3	third report	DeliveredMessagesReport

the energy consumption at each node due to the execution process. Here, we compute the average consumed energy by all nodes. Similarly, *MessageStatsReport* provides some statistics about the different states of the created messages. We use the same report to get the percentage value for successfully executed tasks. Table 4.7 shows the configuration of those report.

4.2.2 Host applications

As mentioned earlier, we develop two applications to handle our scenario. The first application called OffloaderApp, which is hosted at the ofloader nodes. The other application called OffloadeeApp which is hosted at offloadee nodes. Next, we try to describe those applications in details.

4.2.2.1 OffloaderApp

This application module simulates the offloader role in our scenario. The main function of this application is to:

- Handle for created messages; offloaders are message creator, so in the simulation we assign a number of messages for each offloader. Those messages are waiting to be executed in parallel against some potential offloadees.
- Send each message to its potential offloadee and handle for its execution.
- Listen for any ack for the results from offloadees

- Update the energy budget of the offloader
- Report for the delivery time and energy consumption of each task.

4.2.2.2 OffloadeeApp

OffloadeeApp module runs at the offloadee nodes of our scenario. The main function of this application is to:

- Inform the offloader of the availability to receive a task (profile exchange phase).
- Receive the task from the offloader
- Try to execute the task within the current context
- Update the energy budget of the offloadee
- Acknowledge the offloader with the results of execution.

4.2.2.3 TaskApp

This is another application runs at the beginning of the simulation in order to handle for the tasks in our scenario. The main function of *TaskApp* application is to:

- Create a number of messages (tasks) upon specified at the configuration file (i.e. in our scenario from 1 till 6)
- Give each of those messages a required amount for computation as specified in the configuration file (30 MFLOP for medium-computation type and 60 MFLOP for high-computation type)
- Assign the generated tasks to the specified offloaders
- Give the control to original *Message* class in the ONE simulator to control the delivery of each task

4.2.3 Our updates on the ONE simulator

In our simulation scenario, the message creation event is based on a real tracefile (i.e. *Sigcomm09AT*). To allow the ONE simulator to read all information in the same tracefile, we change some core files in the simulator.

4.2.3.1 discovering a bug

Before starting our scenario, we did a sample test for the average contact duration of the Haggle *infocom06* [37] tracefile. Then we build another JAVA custom application outside the ONE simulation to read from the same trace and compute for the average contact duration of the same tracefile. We discover that there is a difference between both average values. After some work within the simulator, we discover that the simulator drops for some reading values from the tracefile after reading 500 entries. We fix this bug in the ONE simulator, and all of our work is on this updated version to guarantee the correctness of generated results.

4.2.3.2 Readable format

Here we convert for the readable format of the ONE simulator to be able to read all information in the same trace. Actually, the default readable dataset format in the ONE simulator is:

timestamp	srcHost	dstHost	UP/DOWN

; where timestamp here represents for uptime or downtime of the contact between source host (srcHost) and destination host (dstHost). The last field is either Up or DOWN. Our work here done by updating the *dieselnetConverter.pl* file which is existing in the toolkit package of the ONE simulator. This file is used as trace converter in the same simulator. We changed that file to be compatible of our format as:

timestamp	srcHost	dstHost	UP/DOWN	commonFriends	commonInterests	friendship

We did that to allow the scenario to read social information provided by *Sigcomm09AT* tracefile in order to be used by some offloading algorithms.

4.2.3.3 DTNHost class

This class represents for either a source host or destination host in the tracefile (i.e. srcHost or dstHost). The source host represents an offloader, while the destination host is an offloadee. To achieve that, we update *DTNHost* class by adding number of variables of type list. This will be considered for offloaders (Group 1 in our scenario). Those lists can be summarized as follows:

- *lstPotentialOffloadees*: a list contains IDs of offloadees currently available within the range of offloader. The format of this list is:

offloadeeId

- *lstFriends*: a list contains the IDs of offloadees that share some common friends with the offloader. The format of this list is:

offloadeeId	friendsNum

- *lstInterests*: another list contains IDs of offloadees have common interests with the offloader. The format for this list is:

offloadeeId	interestsNum

- *lstFriendship*: this list is dedicated for the offloadees having friendship with the offloader (i.e. the value of 1 if this relationship exists, 0 otherwise). The format for this list is:

offloadeeId	friendship

4.2.3.4 Message class

Message class represents for messages created during the scenario's runtime. We update this class to hold a value for execution either locally or at the offloadee side. We did this by adding some variables in the same class. Those values will be modified by the user in the configuration file before running the simulation. Here, we list some of those variables:

- *compVal*: a value for computation of the task. In fact, this value is set in the configuration file as described earlier.

- *engVal*: a value represents for the energy required to execute a task. This value is set from the *TaskApp* application.

4.2.3.5 Metrics

We try to apply the same metrics described in Chapter 3. This is to apply for the same conditions under the ONE simulator.

The total energy of the whole system depends on the Bluetooth scanning energy consumption as described in Table 4.6. Here, we just consider for *scanEnergy* when a task transferred, we ignore for Bluotooth idle state (*baseEnergy*) and response energy (*scanResponseEnergy*).

4.3 Experimental Results

After running our scenario in the ONE simulator, the nodes are moving according to the predefined movement model as shown in Figure 4.3. Moreover, those nodes interact together if they are in the connection range. In this case, the offloader send the task according to the specified criteria and to what we stated in the configuration file. The simulation runs until the *endtime* of the scenario. After that, a set of reports are generated summarizing what happened during the running scenario. In fact, we run our scenario several times and the output is for the average regarding deliver time and consumed energy. We did this against the same set of proposed algorithms as describes in Section 3.1.5.

We test our scenario under specific conditions to ensure that all cases have been tested, we named this as execution scenarios. For that, we define 3 types of those scenarios: (i) local tasks execution at offloaders and (ii) medium-computation task

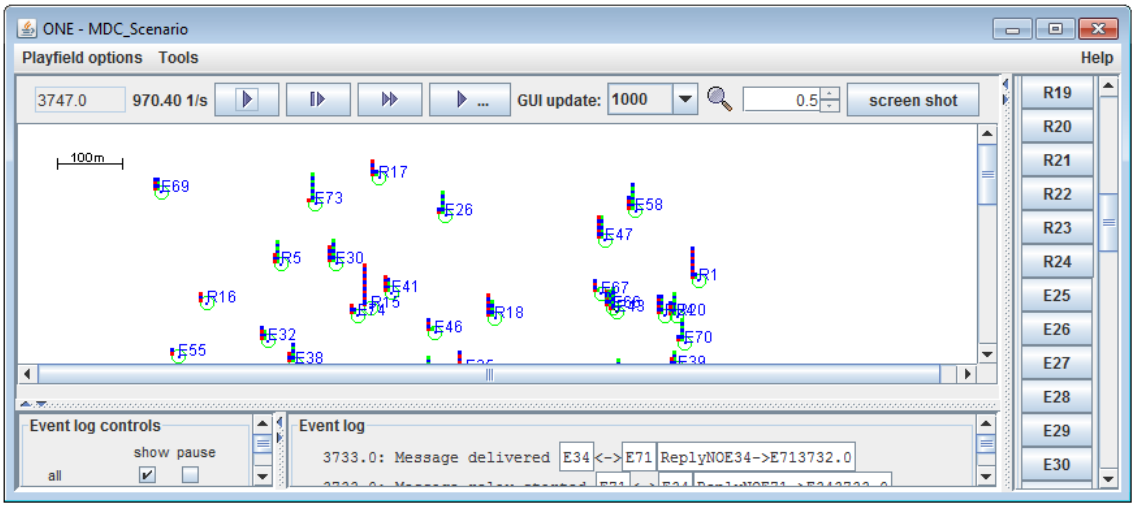

Figure 4.3: View of the running scenario of our work

offloading, and (iii) high-computation task offloading.

4.3.1 Execution scenarios

Here, we describe each scenario briefly. For each offloading scenario, we illustrate how the execution is done based on the parameters mentioned earlier in the configuration file. Moreover, we consider our updates in the ONE simulator environment as described earlier.

In all scenarios, the *TaskApp* application generates number of tasks (u) according to the corresponding values in the configuration file. Those values are:

- *tasksNum*: which represents the number of tasks to be generated at each offloader
- *compVal*: the computation value according to to the type of task (i.e. 30 MFLOP for medium-computation tasks and 60 MFLOP for high-computation tasks)

Here, we fix the task size to be 20 MB as in all scenarios described in the previous chapter. Moreover, all generated tasks are created at timestamp ts_0 which corresponds to the first timestamp in the dataset.

Then, *TaskApp* disseminates the generated tasks to the offloaders. Those tasks are stored temporarily in the buffers provided by each offloader.

4.3.1.1 Local task execution

This is the simplest scenario, here the execution process of all tasks is done at the offloader side. Each offloader has a set of tasks (u) to be executed at each individual offloader. This is actually done through the *Message* class described earlier. In fact, the total execution time and total consumed energy depend on the task type as illustrated in tables 3.4 and 3.5. Then, a set of reports will be generated at the end. We run this scenario separately against the two types of the specified tasks (i.e. medium and high computation tasks). Here, we propose that all tasks are successfully executed on their initiators.

4.3.1.2 Medium-computation task offloading

In this scenario, the offloader send a task of medium-computation type to all potential encountered offloadees according to the offloading criteria (algorithm) described in Section 3.1.5.

4.3.1.3 High-computation tasks offloading

This scenario is identical to the previous one, but here all tasks are of high-computation type.

4.3.2 Execution results

Upon running the scenarios described earlier, we try here to illustrate and discuss about the results for each scenario separately regarding different offloading criteria. We have the same performance metrics described earlier in Section 3.1.2. Those metrics include: delivery time, energy consumption and success rate. Furthermore, we assign a number of tasks (u) to each offloader starting from 1 till 6. Those tasks are created at the same timestamp ts_0, and try to be executed in parallel against a set of potential offloadees available at each offloader. Here, we apply for the two cases regarding the

device profile; i.e. we consider the homogeneous case where all devices have the same profile and heterogeneous case where the offloadees profiles are higher then offloaders.

Homogeneous case We consider here two types of task as described in Chapter 3.

Medium-computation tasks scenario Figures 4.4-4.6 illustrate the results of medium-computation offloading under the homogeneous environment. We can see from Figure 4.4 how the success rate of *S-basedRep* is the highest regarding other offloading protocols and it is very close to the values for the lower-bound (*Flooding*). For both protocols the success rate for $u \leq 4$ exceeds 100%, which means that almost all assigned tasks are successfully executed remotely. This percentage goes a little bit down for $u > 4$, but it is still above 80%. Moreover, *S-based* protocol is not far away from replication version and gives around 80% for $u = 6$. On the other hand, *Random* protocol gives the lowest success rate with around 50% for $u = 6$. Furthermore, *I-based* protocol gives higher success rate compared to *F-based*,but both of them are still below *S-based*.

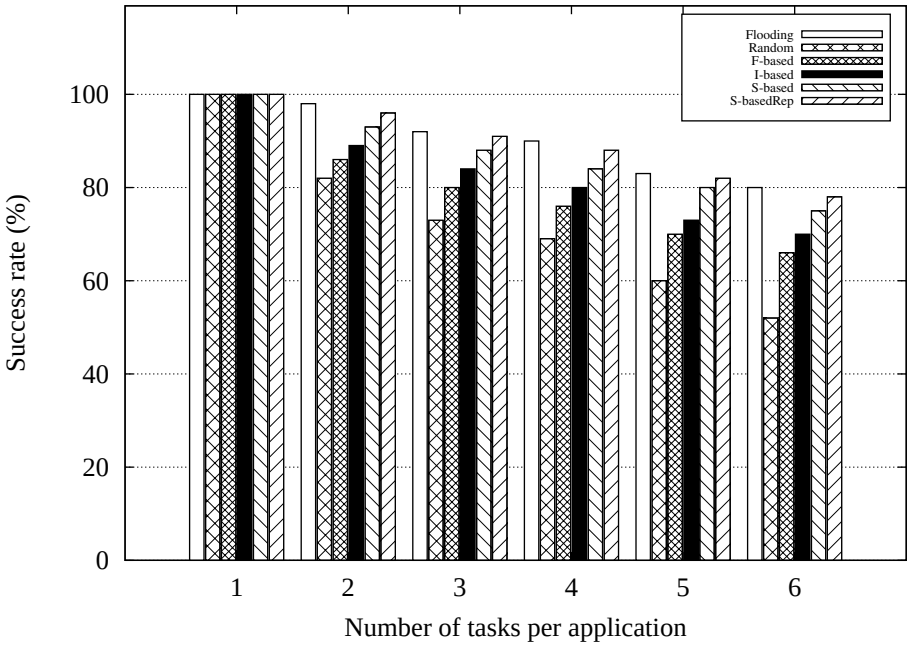

Figure 4.4: Success rate regarding parallel execution of u medium-computation tasks (homogeneous case)

Figure 4.5 plots for the delivery time regarding all set of protocols. In the same figure, we can see how the time for local execution increased dramatically respect to the number of assigned tasks. From the same figure, we show how the delivery time regarding the *Flooding* and *S-basedRep* protocols give the lowest values. *S-basedRep* protocol saves more than 70% of delivery time regarding local execution. In fact, *Flooding* and *S-basedRep* protocols try to send several copies of the same task and wait for the one with shorter execution time. Moreover, *S-based* protocol saves around 55% respect to local execution. However, other offloading criteria save some amount of time but after number of tasks greater than 2. We show this as in *I-based* and *F-based* protocols, which they save around 30% and 25% respectively. Finally, *Random* protocol saves some amount of time with about 12% for number of tasks greater than 3.

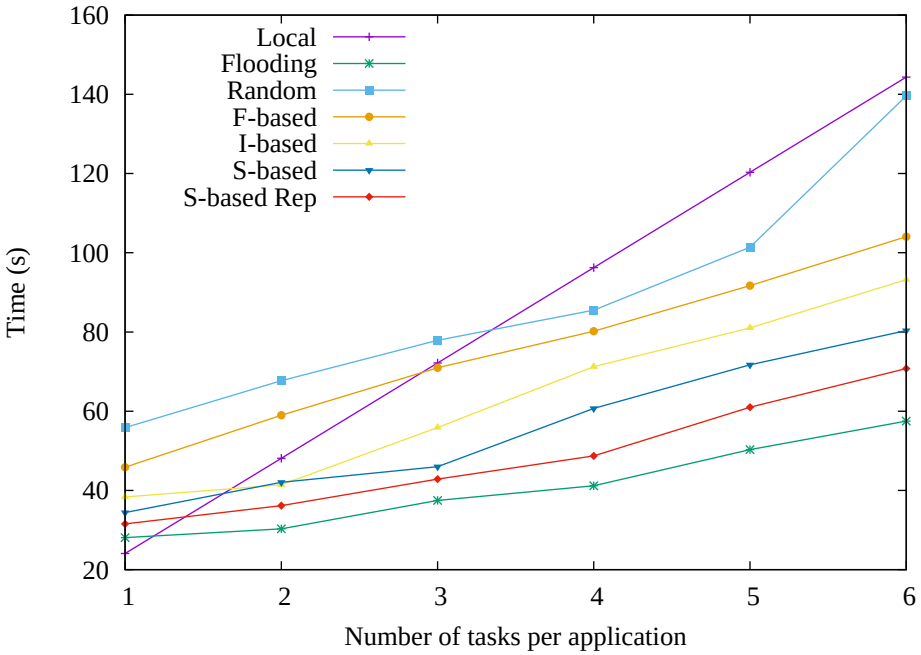

Figure 4.5: Delivery time regarding parallel execution of u medium-computation tasks (homogeneous case)

In Figure 4.6 we can see how the energy consumption regarding all designed protocols. From the same figure, we see how the energy consumption regarding local execution increased in a linear way. Moreover, we notice that *Random* protocol gives

the lowest values for energy with about 65% saving in energy comparing to local execution. Furthermore, *I-based* and *F-based* saves amount of energy equals to about 50% and 45% respectively compared to local execution. Then, *S-based* and *S-basedRep* save about 40% and 35% respectively compared to local execution. Finally, *Flooding* protocol is the worst in energy consumption among all other protocols, but the same protocol still saves around 15% regarding local execution. This can be referred to the amount of energy used to transmit the same task to all available offloadees.

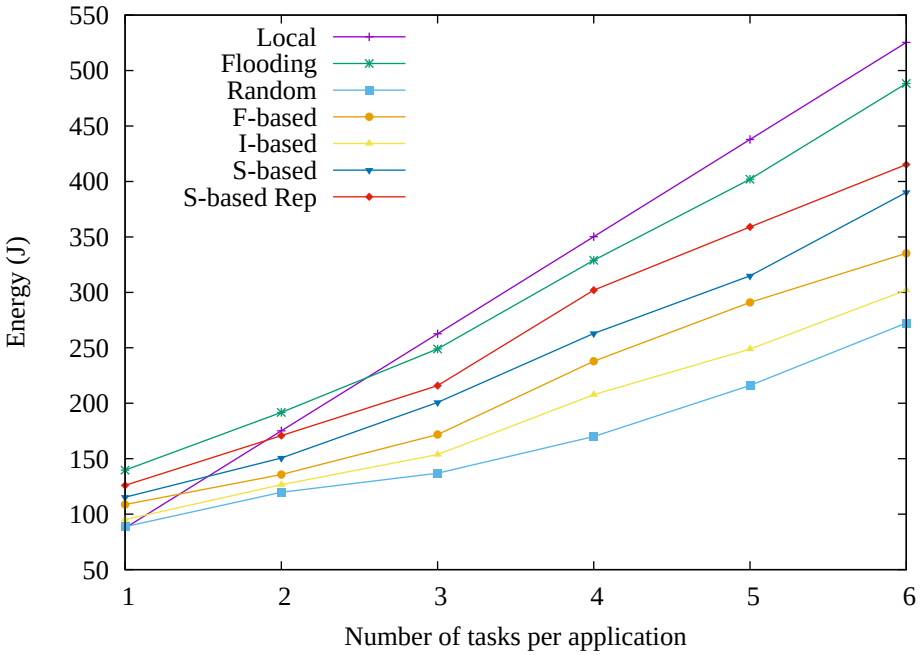

Figure 4.6: Energy consumption regarding parallel execution of u medium-computation tasks (homogeneous case)

High-computation tasks scenario The results of high-computation tasks are illustrated in figures 4.7-4.9. In Figure 4.7, we see how the success rate decreased compared to the same figure for medium-computation offloading. In other words, the number of successfully executed tasks of type high-computation is lower than of type medium-computation. From the same figure, we notice that *Random* protocol gives the lowest success rate with just 45% for $u = 6$. However, *S-basedRep* is still the best strategy

to follow and it is very close to lower-bound. Moreover, *S-based* can be considered the second best strategy here. Other protocols are behave the same, *I-based* gives higher success rate compared to *F-based*.

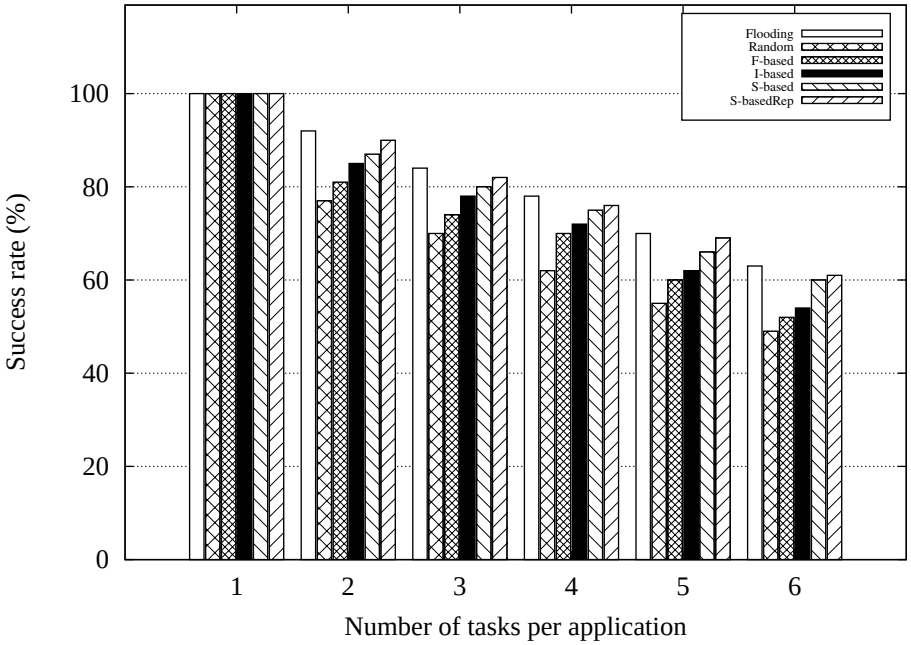

Figure 4.7: Success rate regarding parallel execution of u high-computation tasks (homogeneous case)

Respect to delivery time as shown in Figure 4.8, we still have the same view here. *S-basedRep* gives the lowest delivery time regarding local execution. The same protocol is very close to the lower-bound and it saves more than 50% of time. Furthermore, S-based saves around 40% of time respect to local execution. I-based and F-based save about 25% and 20% respectively. Moreover, *Random* offloading saves some of amount time with about 10%.

In Figure 4.9, we can see how *Random* offloading saves more than 60% of energy respect to local execution. *I-based*, *F-based* and *S-based* protocols are very close to each others and saves around 50% regarding local execution. Furthermore, *S-basedRep* saves around 35% while *Flooding* protocol saves about 25% regarding energy consumption of local execution.

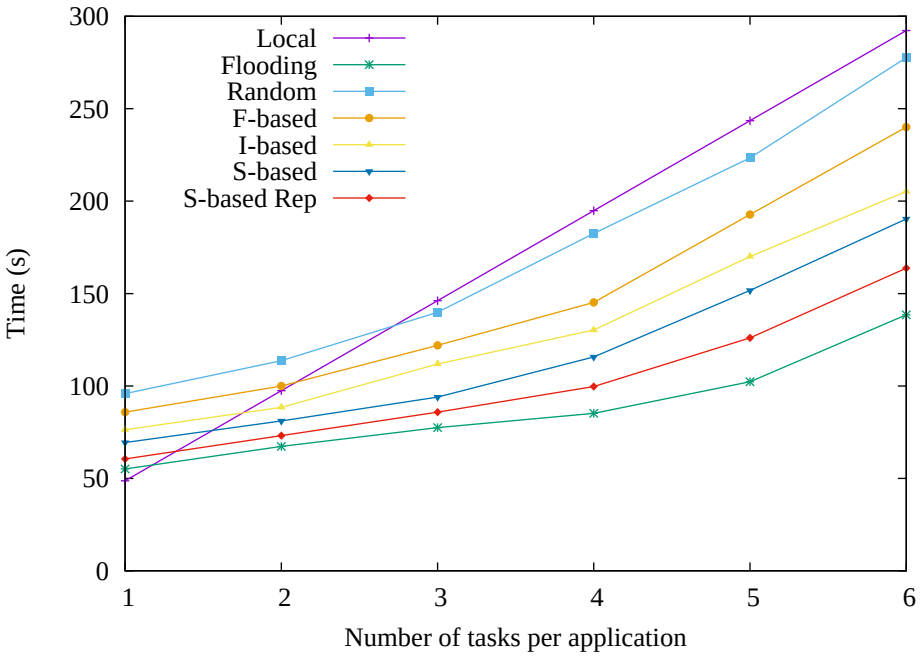

Figure 4.8: Delivery time regarding parallel execution of u high-computation tasks (homogeneous case)

Heterogeneous case We consider here two types of task as described earlier.

Medium-computation tasks scenario Here, we discuss the results regarding heterogeneous case for medium-computation offloading as illustrated in figures 4.10-4.12. We can see from Figure 4.10 how the success rate of *S-basedRep* is the highest regarding other offloading protocols and it is very close to the lower-bound (*Flooding*). For both protocols the success rate for $u \leq 3$ is 100%, which means that all assigned tasks are successfully executed remotely. This percentage goes a little bit down for $u > 3$, but it is still above 96%. Moreover, *S-based* protocol is not far away from replication version and gives around 90% for $u = 6$. On the other hand, *Random* protocol gives the lowest success rate with around 65% for $u = 6$. Furthermore, *I-based* protocol gives higher success rate compared to *F-based*,but both of them are still below *S-based*.

Figure 4.11 plots for the delivery time regarding all set of protocols. In the same figure, we can see how the time for local execution increased dramatically respect to

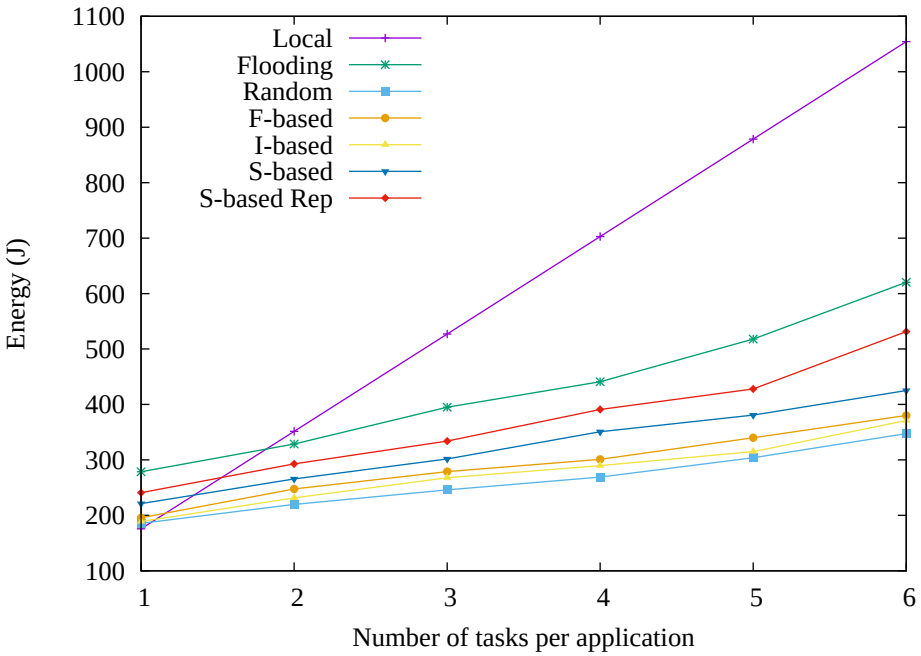

Figure 4.9: Energy consumption regarding parallel execution of u high-computation tasks (homogeneous case)

the number of assigned tasks. From the same figure, we show how the delivery time regarding the *Flooding* and *S-basedRep* protocols give the lowest values. *S-basedRep* protocol saves more than 85% of delivery time regarding local execution. In fact, *Flooding* and *S-basedRep* protocols try to send several copies of the same task and wait for the one with shorter execution time. Moreover, *S-based* protocol saves around 65% respect to local execution. However, other offloading criteria save some amount of time but after number of tasks greater than 2. We show this as in *I-based* and *F-based* protocols, which they save around 40% and 35% respectively. Finally, *Random* protocol saves some amount of time with about 20% for number of tasks greater than 3.

In Figure 4.12 we can see how the energy consumption regarding all designed protocols. From the same figure, we see how the energy consumption regarding local execution increased in a linear way. Moreover, we notice that *Random* protocol gives the lowest values for energy with about 80% saving in energy comparing to local execution. Furthermore, *I-based* and *F-based* saves amount of energy equals to about

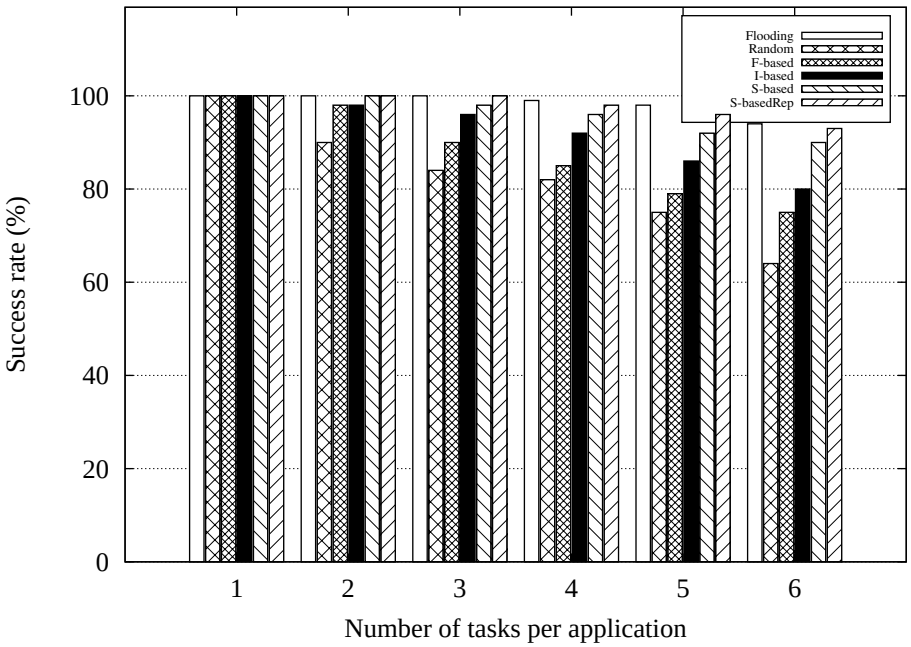

Figure 4.10: Success rate regarding parallel execution of u medium-computation tasks (heterogeneous case)

65% and 60% respectively compared to local execution. Then, *S-based* and *S-basedRep* save about 50% and 45% respectively compared to local execution. Finally, *Flooding* protocol is the worst in energy consumption among all other protocols, but the same protocol still saves around 30% regarding local execution. This can be referred to the amount of energy used to transmit the same task to all available offloadees.

High-computation tasks scenario The results of high-computation tasks under the heterogeneous environment are shown in figures 4.13-4.15. In Figure 4.13, we see how the success rate decreased compared to the same figure for medium-computation offloading. In other words, the number of successfully executed tasks of type high-computation is lower than of type medium-computation. From the same figure, we notice that *Random* protocol gives the lowest success rate with just 60% for $u = 6$. However, *S-basedRep* is still the best strategy to follow and it is very close to lower-bound. Moreover, *S-based* can be considered the second best strategy here. Other

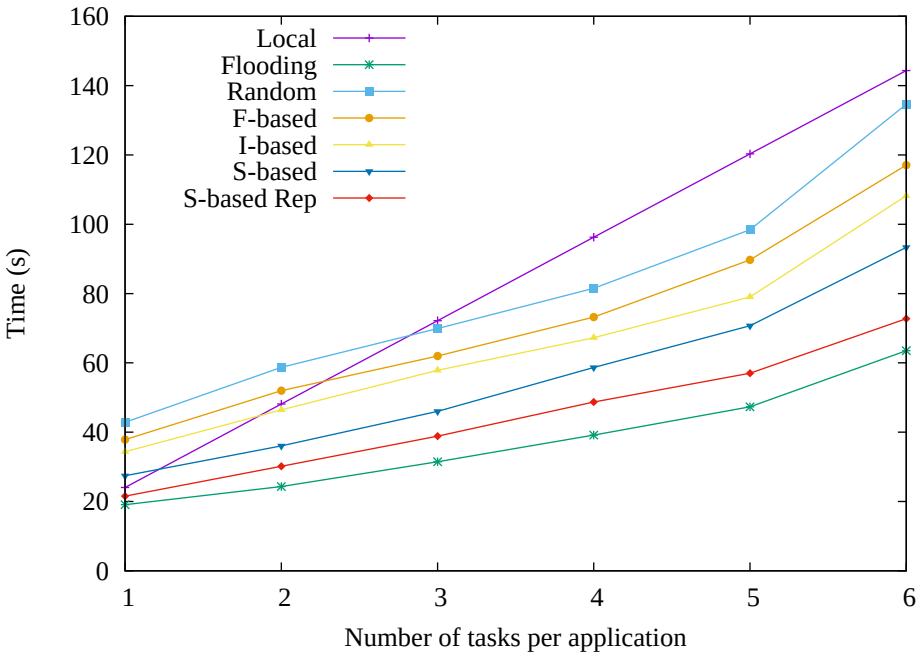

Figure 4.11: Delivery time regarding parallel execution of u medium-computation tasks (heterogeneous case)

protocols are behave the same, *I-based* gives higher success rate compared to *F-based*.

Respect to delivery time as shown in Figure 4.14, we still have the same view here. *S-basedRep* gives the lowest delivery time regarding local execution. The same protocol is very close to the lower-bound and it saves more than 55% of time. Furthermore, S-based saves around 45% of time respect to local execution. I-based and F-based save about 30% and 25% respectively. Moreover, *Random* offloading saves some of amount time with about 15%.

In Figure 4.15, we can see how *Random* offloading saves more than 75% of energy respect to local execution. *I-based*, *F-based* and *S-based* protocols are very close to each others and saves around 60% regarding local execution. Furthermore, *S-basedRep* saves around 45% while *Flooding* protocol saves about 30% regarding energy consumption of local execution.

In a nutshell, our real simulation under the ONE simulator indicates that *S-basedRep* is the best strategy to follow regarding the success rate and delivery time of the assigned

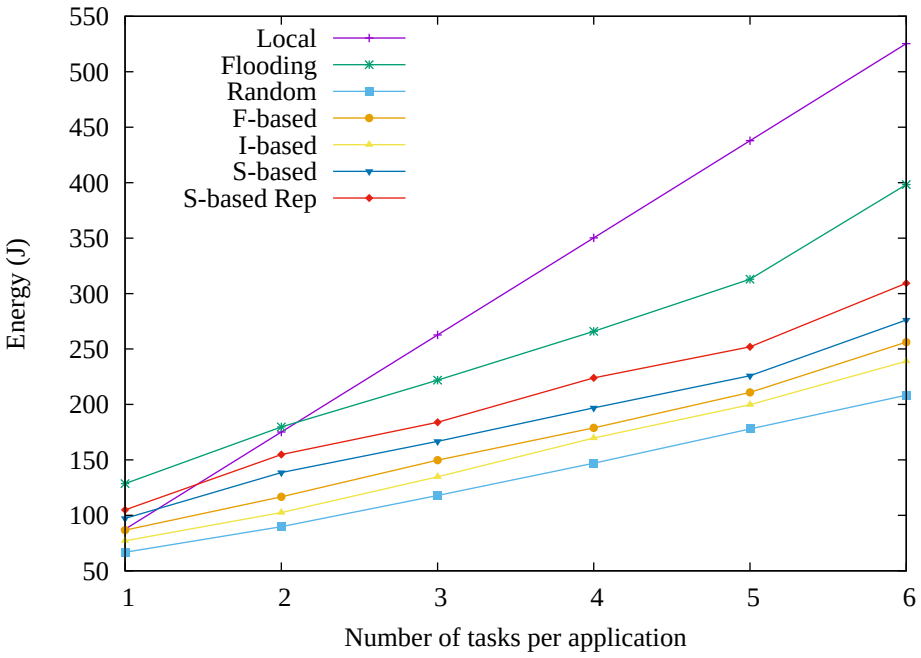

Figure 4.12: Energy Consumption regarding parallel execution of u medium-computation tasks (heterogeneous case)

tasks (u). The same protocol gives acceptable values regarding the energy consumption. Moreover, the second best protocol is *S-based* using the same criteria. In fact, those are the same results that we reached before in numerical simulation as illustrated in Section 3.2.2. The values here are a little bit higher because we consider all network conditions in the real simulation. Those conditions including: collision, packet loss, congestion, etc.

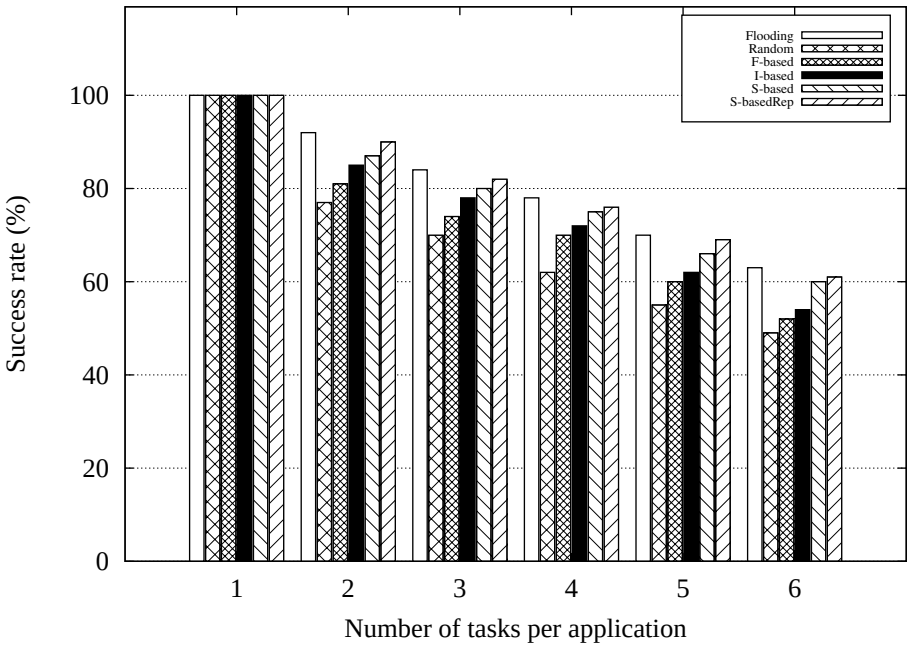

Figure 4.13: Success rate regarding parallel execution of u high-computation tasks (heterogeneous case)

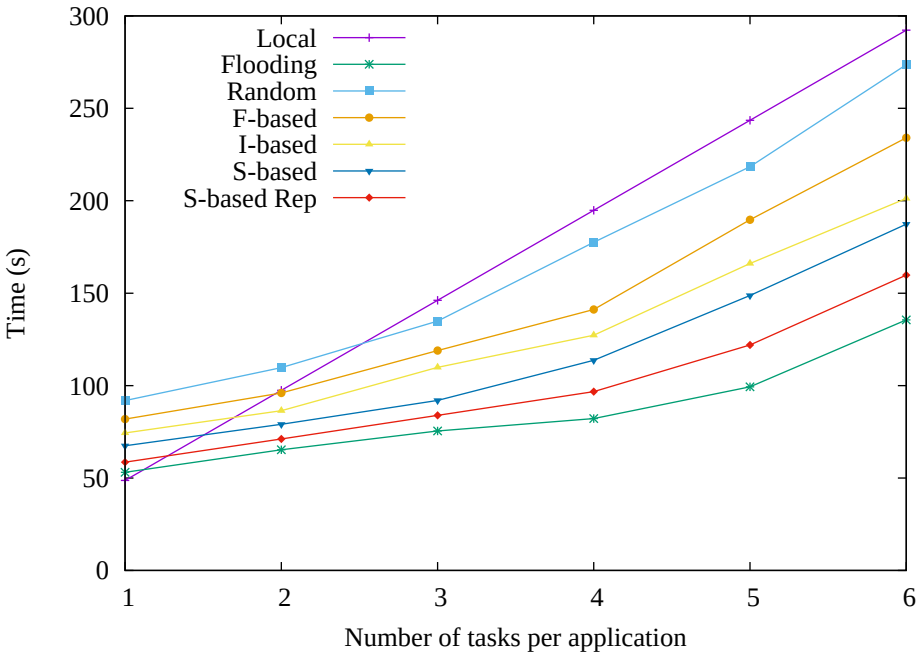

Figure 4.14: Completion time regarding parallel execution of u high-computation tasks (heterogeneous case)

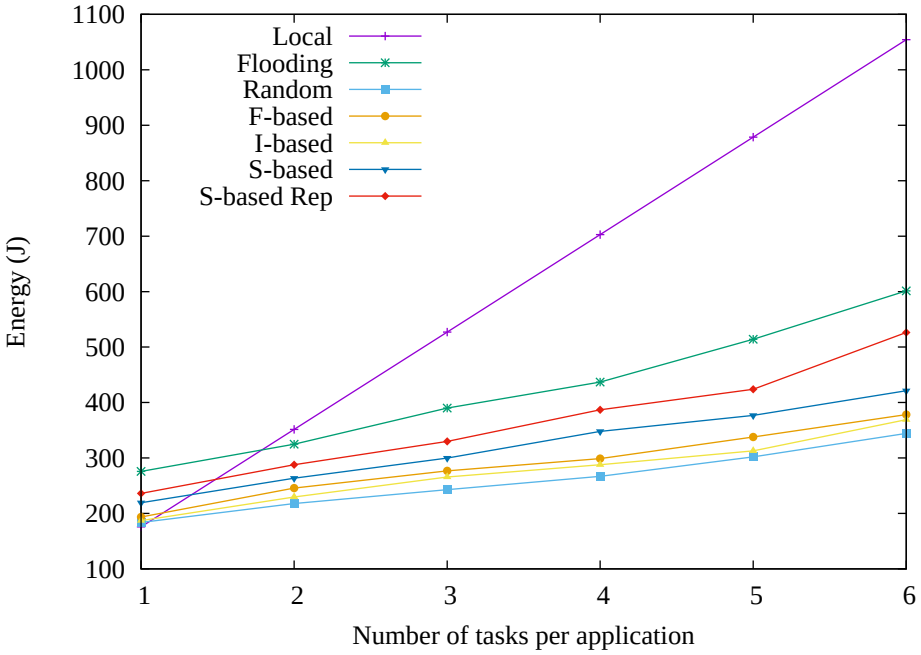

Figure 4.15: Energy Consumption regarding parallel execution of u high-computation tasks (heterogeneous case)

Conclusions and Future Work

Our work starts with a comprehensive investigation study considering some well-known social tracefiles, which they include two basic social factors: friendship and interests. Then, we apply our metrics and constraints in order to quantify for some outcome. After that, we use the outcome from investigation phase to be tested on a custom app in order to test the hypothesis and to measure the gain earned. This app include profiles for real devices and some proposed task capabilities. In this app, we test for two major types of tasks: medium-computation and heavy-computation tasks against different proposed offloading algorithms. Our results from this app indicates that our designed algorithm, which is based on the investigation study, attains for about more than 60% in time regarding local execution for any number of tasks. Moreover, comparing to other offloading scenarios, the same offloading algorithm saves for more than 40% in execution time regarding random offloading. Furthermore, we test for replication factor and we reach for a result indicating that only 2 replicas is enough to overcome the task loss or network failure. Finally, we build another model running in real simulator environment (The ONE simulator) in order to support for our hypothesis and numerical results. Furthermore, the success rate for the same selection criteria gives higher values compared to other offloading with values exceed 90% for different proposed environments (homogeneous and heterogeneous cases).

As future work, we try to find an efficient mechanism for application splitting before offloading code chunks taking into consideration: (i) the tradeoff regarding local

execution and (ii) the remaining offloadee's profile. Moreover, we try to build a movement model considering the social factors between a pair of nodes. This is due to the assumption that nodes share some social aspects will move close to each other. Furthermore, we will work to build a framework running on real mobile devices, and testing for MDC scenario and cloudlet scenario depending on the available available communication spectrum of mobile devices.

Publications

Part of this thesis was published or still working to be published in the following conference and workshop proceedings:

- A. Mtibaa, M. Abu Snober, A. Carelli, R. Beraldi, H. Alnuweiri, "Collaborative Mobile-To-Mobile Computation Offloading," CollaborateCom'14.
- "A trace-based study for social driven D2D code offloading", not completed/ not published yet.

Bibliography

[1] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, "Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications," IEEE, 2013

[2] K. Kumar, J. Liu, YH Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," Springer, 2013

[3] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama and R. Buyya, "Mobile Code Offloading: From Concept to Practice and Beyond", IEEE, 2015

[4] M. Carbone and L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols." ACM, 1997.

[5] B-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," ACM, 2011

[6] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing. Pervasive Computing," IEEE, 2009.

[7] A. Mtibaa, A. Fahim, K. Harras, and M. Ammar, "Towards resource sharing in mobile device clouds: Power balancing across mobile devices," MCC, 2013.

[8] A. Mtibaa, M. Abu Snober, A. Carelli, R. Beraldi, H. Alnuweiri, "Collaborative Mobile-To-Mobile Computation Offloading," IEEE, 2014.

[9] Abolfazli, S., Sanaei, Z., Ahmed, E., Gani, A. & Buyya, R., "Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges," IEEE, 2013

[10] H. Qi, A. Gani., "Research on Mobile Cloud Computing: Review, Trend and Perspectives," IEEE, 2012

[11] M. Khabbaz,C. Assi, and W. Fawaz, "Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges," IEEE, 2012.

[12] A. Voyiatzis, "A Survey of Delay- and Disruption-Tolerant Networking Applications," IEEE, 2012.

[13] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl., "MAUI: making smartphones last longer with code offload," ACM, 2010.

[14] M. Kemppainen, "Mobile Computation Offloading: a Context-driven Approach," Aalto University – Seminar on Internetworking, 2011

[15] K. Habak, M. Ammar, K. Harras and E. Zegura, "FemtoClouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge," IEEE, 2015

[16] C. Shi, P. Pandurangan, K. Ni, J. Yang, M. Ammar, M. Naik and E. Zegura, "IC-Cloud: Computation Offloading to an Intermittently-Connected Cloud, " Georgia Institute of Technology ,2013

[17] C. Wang, Y. Li and D. Jin, "Mobility-Assisted Opportunistic Computation Offloading," IEEE, 2014

[18] M. Barbera, S. Kosta, A. Mei and Julinda Stefa, "To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing," IEEE, 2013

[19] G. Orsinia, D. Badea and W. Lamersdorfa , "Context-Aware Computation Offloading for Mobile Cloud Computing: Requirements Analysis, Survey and Design Guideline," MobiSPC, 2015

[20] A-R. Khan, M. Othman, F. Xia, and A-N. Khan, "Context-Aware Mobile Cloud Computing and Its Challenges," IEEE, 2015

[21] B. Zhou, A-V Dastjerdi, R. Calheiros, S-N Srirama, and Rajkumar Buyya, "A Context Sensitive Offloading Scheme for Mobile Cloud Computing Service," IEEE , 2015

[22] G. Orsini, D. Bade, and W. Lamersdorf, "CloudAware: Towards Context-adaptive Mobile Cloud Computing," IEEE, 2015

[23] R-C. Marin, R-I. Ciobanu, R. Pasea, V. Barosan, M. Costea, and C. Dobre, "Context-Awareness in Opportunistic Mobile Cloud Computing," IGI global, 2015

[24] C. Shi, V. Lakafosis, M. Ammar, and E. Zegura, "Serendipity: enabling remote computing among intermittently connected mobile devices," ACM, 2012.

[25] C. Shi, M. Ammar, E. Zegura, and M. Naik,"Computing in cirrus clouds: the challenge of intermittent connectivity," ACM, 2012.

[26] Traces of Bluetooth encounters, opportunistic messaging, and social profiles of 76 users of MobiClique application at SIGCOMM 2009, http://crawdad.org/thlab/sigcomm2009/20120715/, [last accessed: 10-apr-2017]

[27] Traces of Bluetooth encounters, Facebook friendships and interests of a set of users collected through SocialBlueConn application at University of Calabria, http://crawdad.org/unical/socialblueconn/20150208/, [last accessed: 10-apr-2017]

[28] A. K. Pietiläinen, C. Diot, "Dissemination in Opportunistic Social Networks: The Role of Temporal Communities," ACM, 2012

[29] A. Socievole; F. De Rango; A. Caputo, "Wireless contacts, Facebook friendships and interests: Analysis of a multi-layer social network in an academic environment," IFIP, 2014

[30] Samsung Galaxy and Gear, official website: http://www.samsung.com/galaxy, [last accessed: 12-apr-2017]

[31] A. Fahim, A. Mtibaa, and K. Harras, "Making the case for computational offoading in mobile device clouds," Carnegie Mellon University, 2013

[32] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet, "The linpack benchmark: past, present and future. Concurrency and Computation: Practice and Experience," IEEE, 2002

[33] Bluetooth Technology website, official website: http://www.bluetooth.com, [last accessed: 12-apr-2017]

[34] A.Mtibaa, A. Fahim, K. Harras and M. Ammar, "Towards Resource Sharing in Mobile Device Clouds," ACM, 2013

[35] The One Simulator, official site: http://www.netlab.tkk.fi/tutkimus/dtn/theone/, [last accessed: 17-apr-2017]

[36] A. Keränen, J. Ott and T. Kärkkäinen, "The ONE Simulator for DTN Protocol Evaluation," SIMUTools'09, 2009.

[37] Traces of Bluetooth sighting by groups of users carrying small devices (iMotes) for a number of days, http://crawdad.org/cambridge/haggle/20090529/, [last accessed: 17-apr-2017]

[38] R. Friedman, A. Kogan, and Y. Krivolapov. "On power and throughput tradeoffs of wifi and bluetooth in smartphones," IEEE, 2011