Università degli studi di Padova

Dipartimento di Matematica

Corso di Laurea in Informatica

BlueFall, a swarming protocols testbed: project, use and results

Author:
Matteo Pozza

Supervisor:
Claudio Enrico Palazzi

September 23, 2014

Academic year 2013/2014

Contents

List of Figures

List of Tables

Abstract

The server-client model, which represents the pivot of the greatest part of the networks, starts to waver when the number of active connections to a single server grows in an exponential way. In this context, if the same resource is needed by a group of clients, a possible solution could be that of heaving one of the clients retrieving the resource and then sharing it among the group: the way in which the resource is distributed in the group is called swarming protocol. The research on new swarming protocols generally shows results obtained through simulations: these results could be far from the real performance of the protocols, but tests in a real environment are usually complex and expensive. The software described in this document allows to test swarming protocols in a real environment using wireless short-range protocols which are available in off-the-shelf devices, like mobile phones and laptops.

Chapter 1

Introduction

In recent years the number of devices connected to the Internet has continued to grow, thanks also to the great diffusion of WiFi or 3G/HSDPA enabled devices. In the same way, the number of clients that a generic resource provider has to satisfy simultaneously, has kept increasing. An interesting and representative case study is represented by a popular file that needs to be downloaded by a multitude of clients such as, for instance, an application update. Another classic example is the update just released for some widespread operating system that is made available in a precise and known moment (Ubuntu and iOS operating systems, for example): in this case another important factor is that the resource (or the group of resources) to download could be quite heavy, so satisfying a single client is not a quick operation, worsening the problem. The most immediate solution to this type of problem is the strengthening of the servers through better hardware or software, which is named *vertical scaling*: however the improvements are limited regardless of the goodness of the components used in each single server, so the solution allows only to relieve the problem and its intrinsic limit can be reached very fast. Another typical solution is the *horizontal scaling*: when a certain number of servers is not able to satisfy the entire amount of the clients, adding one or more servers can be the quickest workaround. While this solution is easy and fast, it is also quite expensive since buying or renting a server has a cost that often depends on the number of simultaneous requested connections. Form the server side point of view, the possible workarounds end here.

A possible solution is to partially offload the server load on the clients, making a client acting as a server for other clients. A good use case is the scenario in which a man uses a mobile for family contacts, another mobile for work and a tablet, all with the same operating system (Android, for example), and in a certain moment an update is made available. Through the classic approach, all the devices downloads their own copy of the update, thus leading to multiple, independent and resource consuming download of the same file. Another example is a computer classroom without a solid administration structure: in this scenario, to update all the computers it is necessary to put each of them in downloading updates separately. In this case a smarter solution is to make one device downloading the resource and sharing it with the other two, reducing the number of requested connections to one.

The ways in which a certain number of devices exchange pieces of information in order to obtain the whole information at the end of the sharing process are called *swarming protocols* and they have become an important research topic in recent years, thanks to the development of the ad-hoc networks studies, in particular vehicular

networks ones. In ad-hoc networks the resource provider is usually absent since the information to share are generated by the clients themselves (a sensors network, for example) or it is present for a very short amount of time (the vehicular networks are a good example, since the roadside base station is the resource provider and the vehicles are the clients), so swarming protocols have become a basic need for this type of networks. Researches on swarming protocols can give important results also on the server-client problem exposed since it is necessary a way to make the clients sharing the resource retrieved by one of them: in other words, the clients need a swarming protocol in order that each one of them can obtain the resource. Although the swarming protocols is an active research field, most of the results presented in researches in this topic are based on simulations. This happens mainly for two reasons: the first is that the simulation environment offers the complete control of all the variables involved in the experiments and this grants the absence of external bias in the results, the second one is that experiments in a real environment are usually quite expensive, especially when the study wants to show results on a great number of devices involved. The main problem of the simulation-based results is that they could be quite different from results obtained in a real environment: a trade-off between the quality and the costs of the real environment results is needed.

BlueFall, the software presented in this document, consists in a testbed which implements two swarming protocols and which allows to test them in a real environment through the usage of wireless protocols, like Bluetooth and WiFi-Direct. It allows also to collect various data, like the instantaneous data rate, the time involved and the energy consumption, in order to evaluate the goodness of the considered swarming protocols, and it works on everyday devices, like phones, tablets and laptops. So the aim of BlueFall is dual: from one side it allows to measure the efficiency and the effectiveness of the client-side solution, while from the other side it allows to test swarming protocols using low-end devices, like widespread Android phones.

1.1 Related works

A big number of researches on swarming protocols in wireless environments have been published: their main aim is an efficient (low battery consumption) and an effective (highest data rate and lowest time) data exchange among the devices involved, but almost all of this works present good results working with simulators. The work of Nandan *et al.* [1] presents SPAWN, a vehicular ad-hoc networks protocol in which the vehicles retrieves pieces of information from a base station and they exchange pieces among them in order to obtain the complete resource. The work in [2] tries to get over the known problems of traditional protocols in wireless scenario, like TCP, designing a protocol that takes into account also the devices density. Both works are developed for vehicular networks but their results haven't been applied on a concrete network. In this case it is possible to develop BlueFall implementing the designed protocol: in this way quite good results could be obtained using traditional mobile phones. The work of Yasumoto *et al.* [5] focuses on the energy consumption of the devices involved instead, but also in this work the results are gained through simulations: using BlueFall it is possible to gain more realistic results measuring the battery consumption with hardware tools (the one presented in [30] for example), like in the data collection presented in Chapter 7. Also in the works in [3] and [6] are presented interesting solutions for common swarming protocols problems, like the motion of the devices and the routing of the information: since the results are obtained through the usage

of simulators, BlueFall can be a good basis on which a testing environment can be developed.

1.2 Contributions

The usage of the application presented has given the possibility to perform a data collection and a consequential evaluation of the combinations between swarming protocols and transmission mediums. In particular, the contributions of this work are:

- a study on Bluetooth protocol to share data among 3 and 5 devices, both on mobiles and desktops, including:
  - a comparison of the time involved in sharing between the swarming protocols implemented;
  - a comparison of the average data rate between the swarming protocols implemented;
  - a comparison of the standard deviation of data rate between the swarming protocols implemented;
  - an evaluation of the best Bluetooth-based configuration.
- A study on WiFi-Direct protocol to share data among 3 and 5 devices, both on mobiles and desktops, including:
  - a comparison of the time involved in sharing between the swarming protocols implemented;
  - a comparison of the average data rate between the swarming protocols implemented;
  - a comparison of the standard deviation of data rate between the swarming protocols implemented;
  - an evaluation of the best WiFi-Direct-based configuration.
- an evaluation of the best configuration, both on mobiles and desktops;
- a comparison between the Bluetooth energy consumption and the WiFi-Direct one on mobiles.

All of the contributions can be seen in Chapter 8.

1.3 Outline

The document is divided in chapters which trace the different activities that have been performed during the project:

- Chapter 2 describes the problematic context of the server-client model, then it discusses the state of the art solutions and the idea of BlueFall as an alternative solution;
- Chapter 3 indicates which are the requirements of the software and their classification, then it describes the requirements expansion process and the use cases which have been individuated;

- Chapter 4 reports on the high-level structure of the software, which includes the architectural design pattern and the classes organization in packages and namespaces;
- Chapter 5 describes the most used design patterns, the inner structure of the macro-components and the details of some classes contained in modules;
- Chapter 6 presents the coding process, the problems which have arisen and the solutions adopted, and it describes the development of the documentation;
- Chapter 7 reports on the details of the data collection which has been performed using BlueFall and on the issues encountered using the software;
- Chapter 8 contains the analysis of the data collection results and the hypothesis which have been formulated on the behaviour of the swarming protocols;
- Chapter 9 describes the comparison between the estimation and the final, the considerations on the project and the future works.

Chapter 2

Description of the scenario

The chapter aims to give a detailed description of the context for which BlueFall has been designed: it is explained the problematic scenario presented in Chapter 1, how BlueFall can try to solve the problem, which are the swarming protocols, called here *sharing schemes*, implemented in the application and the pro and cons of the choice of the transmission medium to use in sharings.

2.1 Baseline

The basic scenario is composed of a certain number of elements. They are:

- a server, or resource provider: it is a structure designed to accept requests of owned resources and to satisfy these requests in less time as possible. It is generally designed to support a number of requests greater than 1;
- a certain number of clients, or resource requesters: they are devices that make resource requests to the server and expect to obtain the resources asked. The number of clients in the scenario has to be greater than 1 and the clients are not expected to know the presence of any other client since each client knows only the server to which the resource requests have to be performed;
- a connection between the server and each client that has to perform a resource request allowing the request to reach the server and the resource to be retrieved by the client. It generally consists in an Internet connection, so clients are requested to support a way to connect to the Internet. The way the server is connected to the Internet is not a fundamental detail, but it usually consists in a classic Ethernet cable connection;
- a resource, or a group of resources, that each client has to obtain from the server. The resource corresponds generally to a file.

Figure 2.1 contains all the elements listed in the scenario: the server is represented by the grey database symbol, the clients corresponds to the three laptops A, B and C, and the three blue arrows represent the connections to the server, which is allowing the download of a resource by the clients as suggested by the direction of the arrows.

The problems of this classic scenario start to emerge when at least one of the two following conditions are satisfied:

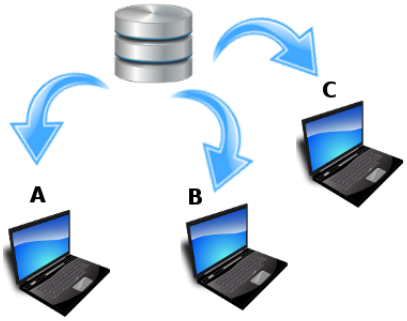


Figure 2.1: The basic scenario

1. the number of clients that are requesting the resource to the server is greater than the number of connection that a single server is able to handle [1];
2. the resource requested from the clients is particularly heavy.

These two conditions are easily satisfied in some common scenarios, like when an update is made available for a popular application or an operating system, as explained in Chapter 1. To solve this problem, two approaches can be adopted:

- a "server side" approach, that solves the problem in a transparent way to the user;
- a "client side" approach, that tries to partially offload the server's burden on the clients.

The server side approach consists in solutions that modify the structure or the number of the resource providers. This type of solutions are not known by the clients that only enjoy the benefits and keep to request resources as before. The most common server side solutions are:

- the vertical scalability, that consists in the strengthening of the single resource provider (substituting hardware pieces or installing better software, for example) in order to support a greater number of incoming connections;
- the horizontal scalability, that consists in the replication of the resource provider structure in order to gain more than one resource provider allowing a distribution of the resource requests among the servers.

The vertical scalability cannot be considered a valid and general solution since it fits only when the number of resource requests is limited: in other words, this solution only shifts the number of clients satisfiable by the server to a greater number, but the vertical scalability intrinsic limit can be easily reached because it is strongly connected with the hardware intrinsic limits. The horizontal scalability represents the most

[1] A server handles correctly a number of connections when it is able to grant a certain quality level of download for each of the connections. The quality level includes a maximum time and a minimum data rate to obtain the resource.

common adopted solution since it is quite easy to add equal resource providers and making them redistributing the resource requests among them. In fact, many vendors offer the possibility to host a resource provider allowing also the horizontal scalability of the server. The biggest problem of this solution is often the price since renting or buying servers may be quite expensive: the prices usually vary on the number of connections requested to the resource provider hosted.

The client side approach consists in solutions that can lighten the work of the server making the clients aware of the presence of other clients with whom sharing the resource. A client side solution needs at least two elements:

1. a way in which the clients exchange pieces of resource in order to obtain the whole resource, that is called swarming protocol;
2. a transmission medium that allows the discover other clients and the communication with them.

The type of the swarming protocol allows to identify two approaches:

- a global approach, in which each client can find every other client that is requesting or offering the resource worldwide;
- a local approach, in which a client knows only a certain number of other clients present in a bounded spacial region.

The global approach needs a common infrastructure among all clients that can be easily identified with the Internet, for example, and the transmission medium usually matches a classic Internet connection (the nature of the transmission medium is not important in this context). A famous swarming protocol that relies on the Internet is BitTorrent and it is adopted as a solution by Canonical Ltd., the Ubuntu developer software company, which encourages the use of the protocol to obtain the releases of the operating system. The local approach doesn't need a pre-existent infrastructure since it is created ad-hoc using the transmission medium, which can correspond to wires or alternatively to wireless mediums.

The work presented in this document belongs to the client side solutions and it corresponds to a local approach. In fact, BlueFall implements two simple swarming protocols that rely on short-range wireless communication protocols, Bluetooth and WiFi-Direct.

2.2 The swarming protocols

Swarming protocols can be designed for a lot of purposes, like routing optimization and energy saving. They are generally created ad-hoc depending on the target network: a swarming protocol for vehicular networks has to be designed taking into account the mobility of the network participants, for example. One of the aims of BlueFall is to allay the server-client model problem explained previously, so that the swarming protocols implemented in it can consider this problem as main objective, also if generally swarming protocols can deal with a very hight number of other problems. The other aim of BlueFall is to become a tool to test swarming protocols: it offers the basic structures to build up a swarming protocol using the local approach.

The two swarming protocols implemented in BlueFall are so simple that can be called sharing schemes. In fact there are no chunks of information exchanged among the devices, on the contrary, the information is spread from a device to a certain number of adjacent devices. The following two sections explain the protocols in detail.

2.2.1 The star sharing scheme

The star sharing scheme has been designed with the aim to re-create the server-client model among the clients alone. A single client requests the resource to the server and it forwards the resource to the other clients that are connected to it, so the client at issue assumes the role of "server-offshoot". Since a single client acts as a resource gateway for all the other clients, the scheme can be considered a centralized sharing scheme.

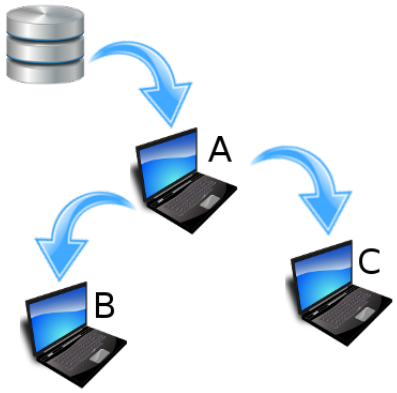


Figure 2.2: The star sharing scheme

Figure 2.2 shows the way of working of the sharing scheme: the device A is the only device that connects directly to the server and which makes a resource request to it. As soon as the resource is being retrieved from the server, A sends the resource to the other clients B and C, till the whole resource has been forwarded. In this way the server connections requested are reduced to one while the clients satisfied are three simply replicating the server-client model among the clients. The scheme is named "star" since the only client connecting to the server can be seen as the center of a star in which every ray is represented by the connection between the client itself and one of the other clients asking for the resource. In this scheme the device which acts as "server-offshoot" is called transmitter, while the clients which are receiving the resource form another client are called receivers: the smartest aspect of this protocol is that the server does not know nothing about the existence of the other clients although they are satisfied in any case. The key aspect of this protocol is that the transmitter client has to satisfy a very limited amount of resource requests, but replicating the protocol application among small groups of devices can reduce the total number of resource requests to the server significantly, as in Figure 2.3.

Since the protocol matches a local approach, a discovery phase is needed: the transmitter puts itself in listening for incoming connections, as a classic server does, while the other clients search for it using the transmission medium. Once the transmitter has been found, the receivers can perform a connection to it.

2.2.2 The waterfall sharing scheme

The star sharing scheme fits very well when the number of receivers is small. In fact if this number is too high the server-client problem is just shifted from the server on

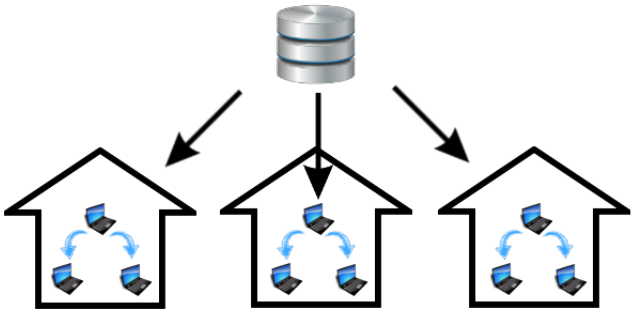


Figure 2.3: Star sharing scheme applied to small groups

the client and since the client's structure is not designed to support a big number of resource requests the problem can only get worse. When the number of clients requesting the same resource is quite high but they are in a limited spatial region (one example is a computer classroom) a better solution may be using the waterfall sharing scheme. In this protocol only one client connects directly to the server and requests for the resource, as in the star sharing scheme, and, as soon as it starts receiving the resource, the client forwards it to just another client that forwards the resource to another client, an so on till the last client, that just receives the resource. In this way each client (except the last one in the chain) acts as a client (to receive the resource) and as a server (to forward the resource) at the same time: since there is not a central node and each client is a peer, the scheme can be considered a distributed sharing scheme. The scheme is named "waterfall" since the resource is transmitted cascade from the server to the last of the clients to satisfy.

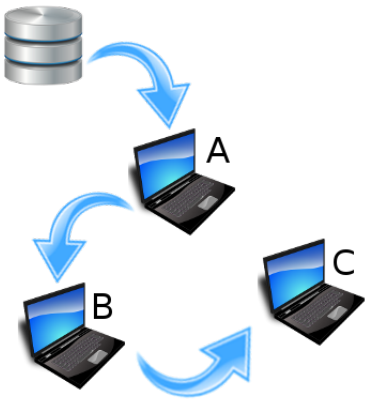


Figure 2.4: The waterfall sharing scheme

Figure 2.4 shows how the protocol works: the client A is connected directly with the server and retrieves the resource but it also forwards the resource to the client B. The client B receives the resource from the client A and forwards the resource to

the client C, that consists in the end of the queue. Since there are no more clients to satisfy, C just receives the resource. The client that retrieves the resource from the server directly is called first node and the client that just receives the resource is called last node, while each client between the first and the last node in the chain is called intermediate node: so, in Figure 2.4, A acts as the first node, B acts as one (and only) intermediate node and C acts as the last node.

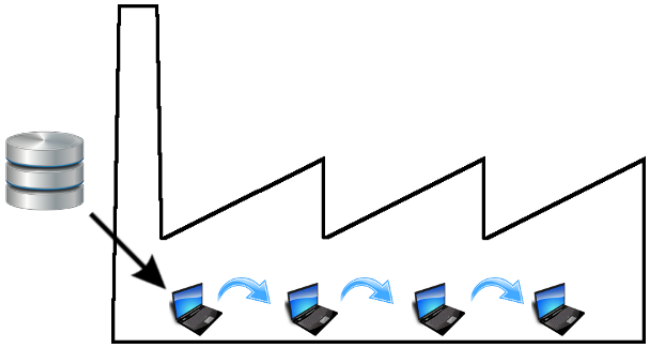


Figure 2.5: Waterfall sharing scheme applied to the computers in a small factory

One of the nice aspects of the protocol is that the server does not know the number of clients satisfied, it just knows the only client that is connected to it, just like in the star sharing scheme. The downside of the protocol is that the entire waterfall structure must be built up before starting to retrieve the resource from the server, so the clients don't fend to understand if another client is already in the waterfall or if they have to assume the last node role. In BlueFall each device does two operations:

- it makes itself available to accept one incoming connection;
- it makes a connection attempt to another client, that must be listening for incoming connections.

The first node in the waterfall only makes itself available to accept an incoming connection, since it has no need to connect to another client (it is already connected to the server to retrieve the resource), while the last node just makes a connection attempt to the penultimate node in the waterfall since there are no other clients that have to make connection attempts to it.

2.3 The transmission medium

The choice of the transmission mediums used by BlueFall was piloted by one of the purposes of the application. In fact, the software needed to be usable on non-expensive devices, so the transmission medium(s) had to be available on almost all the mobiles or desktops.

Wired transmission mediums offer the advantages to grant a very fast data transmission and to set almost to zero the time needed in discovering another device, so they can be considered perfect for a single point-to-point connection. Nevertheless wires don't allow an easy set up of a network without other tools, like routers and switches, since every cable connection needs at least one hardware port in which the cable has

to be put: for example phones have a single cable attachment, so the connections are limited to one single device or to a support structure. Moreover the same type of cable attachment has to be common in all the devices involved and this may result in a big compatibility problem among heterogeneous devices. Also the hindrance of wires has to be considered as a bad aspect of the transmission medium. Finally, the work area is delimited by the length of cables: this may be a problem for devices in different floors of the same building.

The wireless transmission medium has the main advantage that a lot of communication protocols have been developed on it and they are available on a big number of devices, including phones and laptops. These protocols avoid compatibility problems since they are generally designed to be retro-compatible with any previous version and allow to build up a network without the need of any support structure. The bad aspects of the wireless protocols are that data rate is often much lower than the one offered by cable connections and the communications could be easily disturbed with everyday objects, like wireless access points or microwave ovens.

BlueFall uses wireless protocols to apply the sharing schemes explained before: in particular it can use Bluetooth, a low data rate and low energy protocol, and WiFi-Direct, a recent protocol that supports high data rate transmissions. These protocols have been chosen since they are the most common in the latest mobile phones on the market and they are also available on desktops and laptops.

Chapter 3

Analysis

In software production, the phase that follows the initial idea of an application consists in the analysis of the requirements. The requirements allows to understand quickly which are the constraint that the final product must respect and which the final product should respect. In particular, the starting requirements, the ones which are written in the specifications of the software, are called user requirements since they are the direct expression of the stakeholders' will, while the requirements which are expanded from the user ones are called software requirements since they represent the translation of the stakeholders' will. The requisites are particularly important as each module of the software, like each test executed, must be linked with the fulfilment of at least one of the requirements: in this way, to one side each piece of code developed is justified by the presence of one or more requirements, while on the other side it is easy to identify which are the requirements that aren't fulfilled at a certain moment of the development process.

3.1 Classification of the requirements

Since the requirements differ in importance, a classification system is needed. In particular, each requirement is assigned:

- a category,
- a priority.

The category is the expression of the type of the requirement. Four categories can be identified:

- a functional requirement is a requirement that concerns the ability of the software to perform a certain operation. For example, the availability of a certain algorithm is a functional requirement;
- a performance requirement is a requirement that concerns the effectiveness or the efficiency in time and space in which a certain operation is performed. For example, granting the execution of the algorithm in less than a certain amount of time is a performance requirement;
- a quality software is a requirement that concerns the final value of the software in terms of reliability, usability, security and other aspects. For example, granting

that the execution of the algorithm does never allow the application to terminate unexpectedly regardless the input is a quality requirement;

- a constraint requirement is a requirement that concerns the environment in which the software is executed or its developing process. For example, making the algorithm available for a certain platform is a constraint requirement.

The priority is the expression of the importance of the fulfilment of the requirement. Three categories can be identified:

- a mandatory requirement is a requirement that must be necessarily fulfilled. The fulfilment of the requirement is indispensable and compulsory;
- a desirable requirement is a requirement whose fulfilment is not compulsory but adds value to the final product;
- an optional requirement is a requirement whose fulfilment is not important but it may be agreed with the stakeholders.

During the user requirements expansion it is easy to identify requirements that descend from other ones: this father-son relationship can be highlighted through the usage of a hierarchic numerical code. So each requirement has a label that uniquely identify it among the other requirements, and it consist in:

- a "R" letter, that stands for "requirement";
- a letter for the category, among "F" (functional), "P" (performance), "Q" (quality) and "V" (constraint);
- a letter for the priority, among "O" (mandatory), "D" (desirable) and "F" (optional);
- a numeric dotted code, in which each number (except the last one) in the dotted sequence stands for a related requirement, starting from the farther ancestor to the father.

The label allows to facilitate the mapping operations with use cases, modules and tests.

3.2 User requirements

The user requirements represent what the stakeholder expects from the software committed, so the specifications of the software represent a contract that the developers must respect in order to declare the work ended successfully.

The mandatory requirements of BlueFall's specifications are:

1. the application works correctly on Android 4.0 ICS - API level 14;
2. the application allows the download of a file from Internet;
3. the application allows the share of a file through Bluetooth protocol;
4. the application allows the share of a file through WiFi-Direct protocol;
5. the application implements the centralized "star" sharing scheme;

6. the application implements the distributed “waterfall” sharing scheme;
7. the application provides a graphical interface for the download and the share of files;
8. the application provides the instantaneous data rate measure of the incoming connection (the connection to the source of data);
9. the application provides the average data rate measure of the the incoming connection (the connection to the source of data);
10. the application provides the data rate's standard deviation measure of the incoming connection (the connection to the source of data);
11. the application allows to generate a log file of the sharing performed;
12. the application allows to choose which data have to be stored in the log file;
13. the application allows to choose how much data have to be stored in the log file (the collection frequency);
14. the application's manual and documentation to install, use and maintain the testbed are provided.

The desirable requirements of BlueFall's specifications are:

1. the application allows the share of a file in the file system;
2. the application allows the download of a sequence/group of files from Internet;
3. the application provides the instantaneous data rate measure of the outcoming connection (the connection to the destination of data);
4. the application provides the average data rate measure of the the outcoming connection (the connection to the destination of data);
5. the application provides the data rate's standard deviation measure of the outcoming connection (the connection to the destination of data);
6. the application's manual and documentation are provided in English.

The optional requirements of BlueFall's specifications are:

1. the application works correctly on Linux Ubuntu 14.04 LTS;
2. the application provides the possibility to create networks with both mobiles and desktops/laptops;
3. the application provides a check to advertise the user if Bluetooth and WiFi-Direct are not supported by the device;
4. the application allows to choose the unit of measure used in displaying and storing data collected (only of data relating the data rate);
5. the application allows to choose the format of the log file created during share.

All the user requirements have been satisfied, except for the optional requirement that asks for the possibility to create mixed networks, so the state of the art of the application allows networks of Android devices only or desktops/laptops only.

3.3 Software requirements

The software requirements consists in the expansion performed by the software team in order to gain atomic and easy verifiable requirements starting from the user ones. In fact, user requirements are generally few and stout since a lot of requirements are implicitly asked.

Each requirement must be mapped to a source since there cannot be unjustified requirements into the project. In this case each requirement is mapped with the specifications of the software since they consist in the only source available.

The following tables contains all the requirements obtained after the decomposition process: each requirement has its label, description and source. Since no performance requirements have been identified, the performance requirements table is absent.

3.3.1 Functional requirements

Requirement ID	Description
RFO1	The application allows the sharing of a file.
RFO1.1	The application allows the download of the file to share from Internet.
RFD1.2	The application allows the usage of a file from the storage memory for the sharing.
RFO1.3	The application allows the sharing of a file through Bluetooth.
RFO1.4	The application allows the sharing of a file through WiFi-Direct.
RFO2	The application implements the “star” centralized sharing scheme.
RFO2.1	The application implements the “star” centralized sharing scheme using Bluetooth.
RFO2.2	The application implements the “star” centralized sharing scheme using WiFi-Direct.
RFO3	The application implements the “waterfall” distributed sharing scheme.
RFO3.1	The application implements the “waterfall” distributed sharing scheme using Bluetooth.
RFO3.2	The application implements the “waterfall” distributed sharing scheme using WiFi-Direct.
RFO4	The application offers a graphical interface to the user.
RFO4.1	The graphical interface allows to choose the transmission medium to use (Bluetooth/WiFi-Direct).
RFO4.2	The graphical interface allows to choose the sharing scheme to use in the transmission.
RFO4.3	The graphical interface allows to choose the role to play in the sharing scheme.
RFO4.3.1	Chosen the “star” centralized sharing scheme, the graphical interface allows to choose the transmitter role.
RFO4.3.2	Chosen the “star” centralized sharing scheme, the graphical interface allows to choose the receiver role.
RFO4.3.3	Chosen the “waterfall” distributed sharing scheme, the application allows to choose the first node role.

RFO4.3.4	Chosen the "waterfall" distributed sharing scheme, the graphical interface allows to choose the intermediate node role.
RFO4.3.5	Chosen the "waterfall" distributed sharing scheme, the graphical interface allows to choose the last node role.
RFO4.4	The graphical interface allows the choice of the resources to share.
RFD4.4.1	In case of local resource sharing, the graphical interface allows to choose a file from the device file system.
RFO4.4.2	In case of remote resource sharing, the graphical interface allows to insert the URL of the file to share.
RFD4.4.3	In case of multiple local resources sharing, the graphical interface allows to choose the file representing the local files.
RFD4.4.4	In case of multiple remote resources sharing, the graphical interface allows to choose the file representing the remote files.
RFO4.5	The graphical interface allows the choice of the partner/s to connect to.
RFO4.6	The graphical interface allows the choice of the log file configuration.
RFO4.7	The graphical interface shows the data relating to the ongoing sharing.
RFO4.7.1	The graphical interface shows the total time involved in resource fetching and sharing.
RFO4.7.2	The graphical interface shows the data relating to the incoming connection.
RFO4.7.2.1	The graphical interface shows the percentage of completion in resource fetching.
RFO4.7.2.2	The graphical interface shows the instantaneous data rate of the incoming connection.
RFO4.7.2.3	The graphical interface shows the average data rate of the incoming connection.
RFO4.7.2.4	The graphical interface shows the standard deviation of data rate of the incoming connection.
RFD4.7.3	The graphical interface shows the data relating to the outcoming connection.
RFD4.7.3.1	The graphical interface shows the percentage of completion in resource transmission.
RFD4.7.3.2	The graphical interface shows the instantaneous data rate of the outcoming connection.
RFD4.7.3.3	The graphical interface shows the average data rate of the outcoming connection.
RFD4.7.3.4	The graphical interface shows the standard deviation of data rate of the outcoming connection.
RFD4.7.4	The graphical interface shows the number of the resource being shared over the total number of resources to be shared.
RFO4.8	The graphical interface allows to set the maximum number of bytes for a single transmission (single socket transfer).
RFD5	The application allows the sharing of multiple files.

RFD5.1	The application allows the download of the files to share from Internet.
RFD5.2	The application allows the usage of multiple files in the storage memory of the device for the sharing.
RFD5.3	The application allows the sharing of multiple resources through Bluetooth.
RFD5.4	The application allows the sharing of multiple resources through WiFi-Direct.
RFO6	The application generates a log file relating to the last sharing carried out.
RFO6.1	The application allows to choose which data has to be saved in the log file.
RFO6.1.1	The application allows to save the data relating to the incoming connection in the log file.
RFO6.1.2	The application allows to save the data relating to the outcoming connection in the log file.
RFO6.1.3	The application allows to save the data relating to the energy consumption in the log file.
RFO6.2	The application allows to choose how much data has to be collected in the log file (the collection frequency).
RFD6.3	The application allows to choose whether the log file has to be saved or not.
RFF6.4	The application allows to choose the log file format.
RFF6.4.1	The application allows to save the log file in the XLS file format.
RFF6.4.2	The application allows to save the log file in the XML file format.
RFF7	The application allows to choose the unit of measure which is used to show and to store the sharing data.
RFF7.1	The application uses the bit-based units of measure (kbps, Mbps, Gbps).
RFF7.2	The application uses the byte-based units of measure (kBps, MBps, GBps).
RFF8	The application works with different type of devices.
RFF8.1	The application works in a mixed devices network (mobiles and desktops).

Table 3.1: The functional requirements table

3.3.2 Quality requirements

Requirement ID	**Description**
RQO1	It is provided the documentation of the application.
RQO1.1	It is provided the documentation to install the testbed.
RQO1.2	It is provided the documentation to use correctly the testbed.
RQO1.3	It is provided the documentation to maintain the testbed.
RQD1.4	The documentation is written in English.
RQF1.5	It is provided the documentation of the source code.

Table 3.2: The quality requirements table

3.3.3 Constraint requirements

Requirement ID	Description
RVO1	The application works properly on Android 4.0 ICS (API level 14)
RVF2	The application works properly on Linux Ubuntu 14.04 x32.
RVF3	The application works properly on Linux Ubuntu 14.04 x64.

Table 3.3: The constraint requirements table

3.4 Use cases

The use cases are scenarios in which the software can be used to perform a certain operation. These scenarios are derivable from the software specifications, so the use case identification process is very important since it can help to identify new requirements that were not discovered during the user requirements expansion. In the same way, once all possible interactions of the user have been identified, the requirements that do not match any of the use cases could be considered superfluous. A helpful tool that helps to make a use case more understandable is the UML diagram: in fact, texts are often sources of ambiguity while schemes written using a standard language avoid this complication.

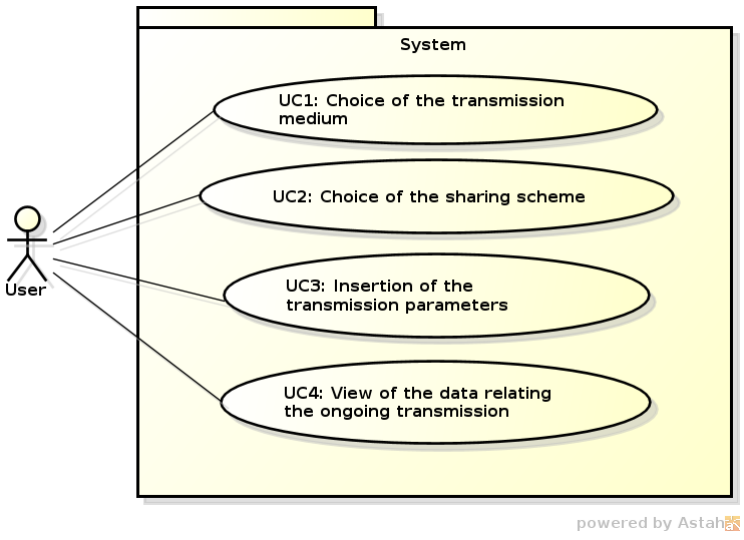


Figure 3.1: Use case UCP - Main stream

Here are presented two use cases that are particularly interesting. The one presented in Figure 3.1 consists in the main use case of the software since it shows the functionalities that the software provides to a user that has just started the application, and these are:

- the configuration of the parameters relating the sharing (UC1, UC2 and UC3);
- the visualization of the data collected by the software during the sharing (UC4).

In particular, the configuration of the parameters can be divided into three sub-functionalities offered by the system, which are:

- the configuration of the transmission medium, that consists in the choice of the wireless protocol to be used in sharing (UC1);
- the choice of the sharing scheme (UC2);
- the configuration of the other parameters useful for the definition of the sharing (UC3).

The first two functionalities have been separated by the generic "configuration of the parameters" since the number and the nature of the parameters that have to be configured depends on the first two parameters (the protocol and the sharing scheme). The preconditions of this use case are not so many: the only one is that the application has just been started correctly. Moreover there is only one actor which interacts with the system, that is the application user. The postconditions of the use case are few, since after the user has participated to the use case, the possible scenarios are 3:

- the user has ended his sharing successfully;
- the user has exited from the application during the configuration of the parameters;
- the user has exited from the application during the sharing.

The main scenario of the use case is quite simple:

1. the user chooses the protocol to be used in sharing;
2. the user chooses the sharing scheme to be used in sharing;
3. the user configures the other parameters of the sharing;
4. the user visualizes the data collected during the sharing on the display.

There are no alternative scenarios since the only other possibility that the user has is to exit from the application during one of the steps.

The other interesting use case is the one in Figure 3.2: it consists in the expansion of UC3 already presented in UCP. In the configuration of the other parameters, the system provides these functionalities to the final user:

- the possibility to set the maximum number of bytes in a single socket transmission (a sort of "high level" packet size) (UC3.1);
- the possibility to choose what has to be saved in the log file (UC3.2);
- the possibility to choose how much data has to be saved in the log file (how much times the sharing should be sampled) (UC3.3);
- the possibility to choose the log file format (UC3.4);
- the possibility to choose the unit of measure with which data have to be shown and to be saved (UC3.5);

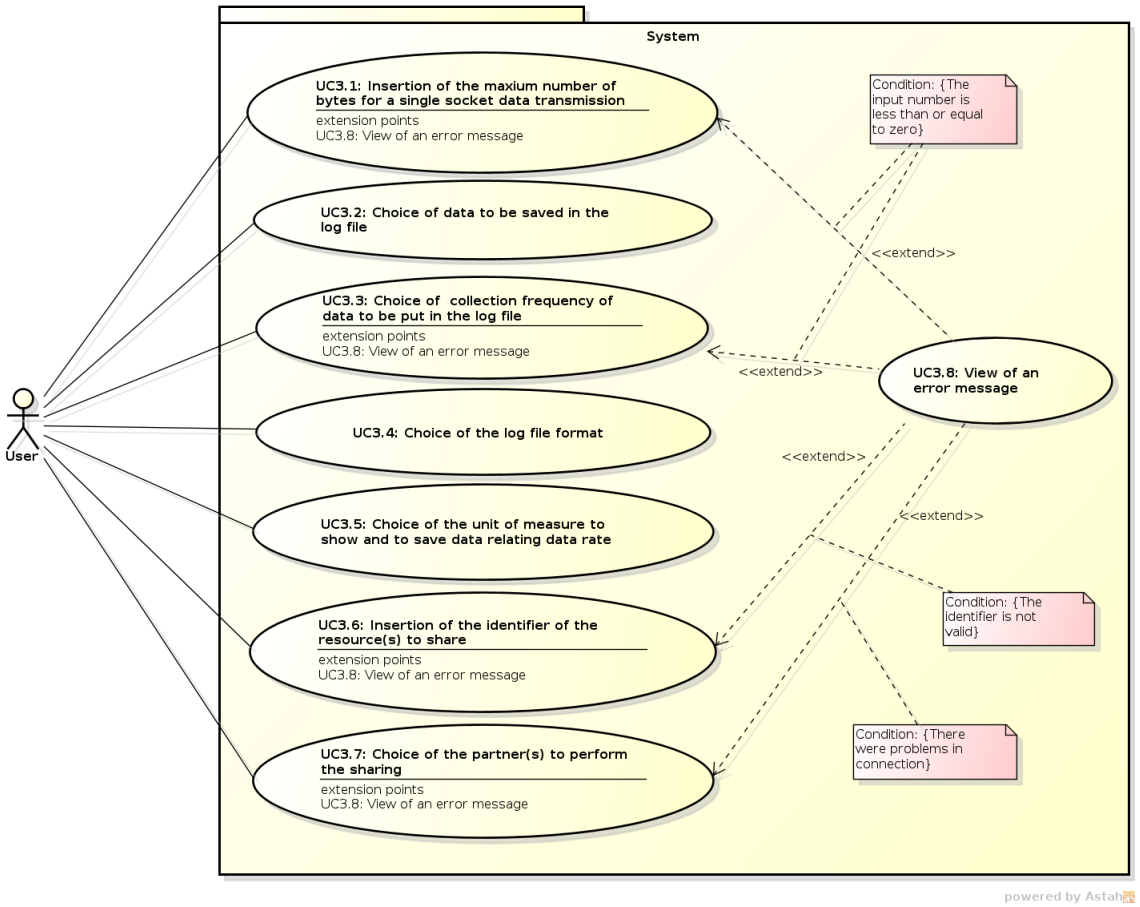


Figure 3.2: Use case UC3 - Insertion of the other parameters

- the possibility to insert the resource(s) identifier(s) that match the file(s) to be shared (UC3.6);

- the possibility to choose the partner(s) in sharing (UC3.7).

Most of these functionalities are not just possibilities but they are choices that have to be mandatory done: for example, it is not possible to start the sharing if no resources to share have been set or if no partners have been chosen. Some of these use cases have an extension point, that is another use case to which the user is brought if certain conditions happen, brusquely exiting from the use case being performed. In fact, the insertion of a number less than or equal to zero, like data collection frequency for example, corresponds to a deviation from the normal stream of the application since it matches an error situation: if it happens, the system answers displaying an error message, which corresponds to UC3.8. The preconditions of this use case are having chosen the protocol and having chosen the sharing scheme to use in transmission, while the postcondition consists in having correctly set all the parameters corresponding the functionalities offered by the use case. The only actor which participates to the use case is the user of the application.

3.5 Validation and system tests

The validation tests consist in procedures that can be performed during the final test of the application: their aim is to grant that the user requirements are fulfilled by the software produced. On the other hand, the system tests aim to grant the fulfilment of the software requirements: they are more specific and they don't consist in procedures to be executed, but they are designed and used by the software developers. It is possible to identify three stages in BlueFall's way of working:

1. the first step is the configuration of all the parameters that concern the sharing, like the transmission medium to use, the sharing scheme chosen and others;
2. the second step is the execution of the sharing;
3. the third step is the check of the integrity of any file received and of the log file generated performed by the user.

The tests have been designed taking into account these three stages. In particular:

- the tests that concern the first stage check that the configuration designed by the user through the graphical interface matches the system's one;
- the tests that concern the second stage check that the sharing proceeds normally, unless it is stopped by the user himself and that the data shown during the sharing are consistent;
- the tests that concern the third stage check the consistency of any resource received and that the log file generated respects the specifications of the user.

Since validation tests and some system tests consist in typical executions of the software each one of the stages is considered, while other system tests focus only on one of the stages.

Chapter 4

Architectural design

Once all the main requirements have been identified the step that follows is the software design. In particular, design has to be divided in two stages: the architectural design, that builds up the scaffolding of the application, and the detailed design, that consists in a finer grained modelling of the software. The architectural design generally starts with the choice of an architectural design pattern, that consists in a valid base on which modelling can start since it is a solid and tested solution to classic design problems, then the identification of the high level components follows.

4.1 The architectural design pattern

The most common architectural design pattern for a GUI-application is certainly the MVC. This pattern describes the separation of three macro-components:

- the model, that includes the business logic of the application;
- the view, that consists in the structures that are presented to the final user;
- the controller, that contains the modules that drive the application and which update the model depending on the user interactions.

Figure 4.1 contains the typical MVC diagram: the green arrow that starts from the view and which goes to the model represents the ability of the view to update itself by reading the updates of the model, the red arrow from the controller to the model represents the changes performed by the controller on the model, while the beige arrow from the controller to the view represents the ability of the controller to change the view depending on the user interactions.

MVC has been adopted as the architectural design pattern for the application since it fits quite well: however some light modifications have been applied in order to find a good compromise between the design pattern and the development platforms used. In particular, model doesn't communicate directly with any other component (view or controller), but it sends broadcast notifications at most: the model is totally independent from the other modules and the controller and the view can use its components as "passive" tools. In this way if the view or the controller must be heavily modified, it will not be necessary to modify the model too.

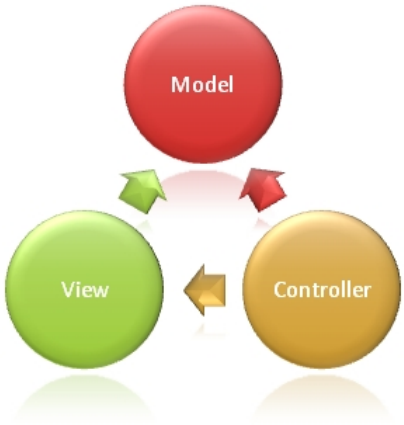


Figure 4.1: The MVC diagram

4.1.1 MVC applied on Android platform

Applying the MVC to an Android application is not a simple job since the Android framework [10] provides a precise model to follow in developing an application. The fundamental unit of an Android application is the single screen of the application and each screen must be handled by a specific component, designed to control the user interactions and the visualization of the graphic elements: each one of these components is named "activity". So an Android application can be clumsily seen as a sequence of activities, each one handling a screen reachable by user interactions.

The activity structure follows quite well the aims of the controller: in fact, it watches the user interactions on the screen and acts accordingly. The biggest problem is that in Android applications the activities are engaged in showing and handling also the elements of the graphic interface. In fact, Android applications have a dual graphic interface management:

- the definition of the graphic interface (what appears and where appears) is made through XML files;
- the handling of the graphic interface (the evaluation of the user inputs and the changes that have to be performed) is made through activities.

In this way controller and view aims meet together making very hard the separation of the components. The solution adopted is to avoid the creation of a separate view since the set of the XML files is already separated from the normal code and can be seen as the "pure" view of the application. Actually, the controller is mainly composed by activities that both modify the model and handle the view. The model is quite easy to identify: it consists in a set of structures that are used by the activities when needed. Accordingly to the preferences about the model expressed previously, the model should not contain modules that are concerned with the activity structure: in fact a model component that is concerned with activities structures means that the model is depending on the controller contents.

4.1.2 MVC applied on desktop platform

The choice of the programming language to use in developing the application is a decision that should be done after the design since one of its aims is to be almost language independent. However in this case the need to work with Bluetooth and WiFi-Direct has driven the choice as there are few libraries that are able to work with these two protocols, especially with the second one. These libraries have been developed in C: since an object-oriented and a C-compatible programming language is needed, C++ is the language used in developing BlueFall for desktops.

The framework used to develop BlueFall is Qt [11] since it is one of the most easy and complete frameworks available for C++ language applications. Qt offers a XML layout definition like Android framework does but it does not force the layout handling in a specific structure: this flexibility allows to separate the structures that handles the user inputs from the ones that modify the model accordingly. So the desktop version of BlueFall has 3 separate components, as described from the MVC pattern, and the layout definitions (the "pure" view) belongs to the view components, together with the structures that handle the layouts.

Another motivation that leads to the choice of Qt is that it offers a particular way that modules can use to communicate, the "Signal-Slot" mechanism: it allows two components to communicate keeping a loose coupling among them. The mechanism is heavily used when the model, the view and the controller have to communicate between them, granting a clean separation of the three components. More details on the mechanism and on how it has been used in the project are available in [18].

The application structure as a sequence of activities imposed by Android platform fits quite well for the purposes of the software since some steps have to be executed sequentially (choice of the protocol, choice of the sharing scheme, parameters definition and so on). In the desktop version of the application it has tried to emulate the same structure of the mobile one: the Qt framework provides a structure that fits the need of a sequence of steps called "QWizard", which consists in a sequence of pages that can be visited sequentially in order to perform a complex work.

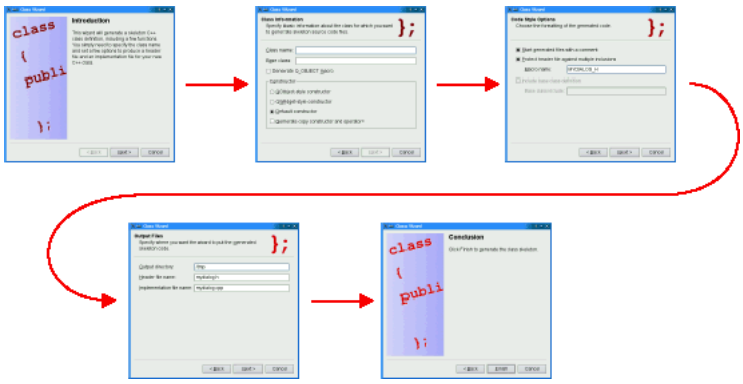


Figure 4.2: The QWizard structure

The QWizard structure is composed by a page handler, the "QWizard" itself, and the pages to be visited, which are represented by "QWizardPage" objects. In order to fit the structure provided by Qt with the MVC pattern, each QWizardPage is considered a view component that handles a single screen of the application, so a specific controller component is linked with each QWizardPage defined. While in the

Android application the view and the controller of each screen are forced to coexist in the same object (a single activity), in the desktop version of the application each view handler is connected with its own controller component that takes care to do the job which depends on the user input.

4.2 High level components identification

The high level components are the structures that separate code with different purposes. Generally these components are organized in a Chinese boxes structure: each component can contain many other components, depending on the aims of the code in them. The MVC pattern provides a base on which other high level components can be designed: in fact model, view and controller can be seen as the biggest containers for other components since any other code container should be part of one of them. Since the two versions of the application differs only for the platform, the Android version and the desktop version share a big number of components with the same name and the same purposes.

4.2.1 Components on Android platform

Since the language used to develop the application is Java, code has to be separated into packages. All the packages are contained into the global application package named "bluefall": in particular, its two direct sons packages are the "controller" package and the "model" one. Since the view components are represented only by XML files, a "view" package is absent. Moreover, the global package should contain the classes that each structure in the sons packages might need: these are the classes that help to implement a design pattern, for example. Figure 4.3 represents the inner structure of the global package: it can be seen that it contains the classes to implement the Observer design pattern.

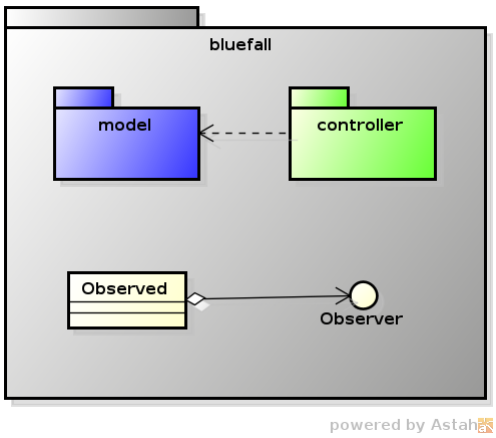


Figure 4.3: The "bluefall" package scheme (mobile)

The "model" package contains a lot of sub-packages in which the structures are put depending on their purpose. In order to avoid to leave any structure used by many sub-packages in the global model package, a dedicated sub-package has been designed. The contents of the model package are:

- the “connect” package, that contains the classes engaged in transmitting and receiving data from other devices;
- the “core” package, that contains the classes engaged in storing the parameters given by the user;
- the “energy” package, that contains the structures to measure power consumption;
- the “log” package, that contains the structures engaged in creating, writing and saving log files;
- the “read” package, that contains the classes engaged in reading data from a source;
- the “save” package, that contains the classes engaged in saving data to a destination;
- the “utility” package, that contains structures used by the other packages’ classes.

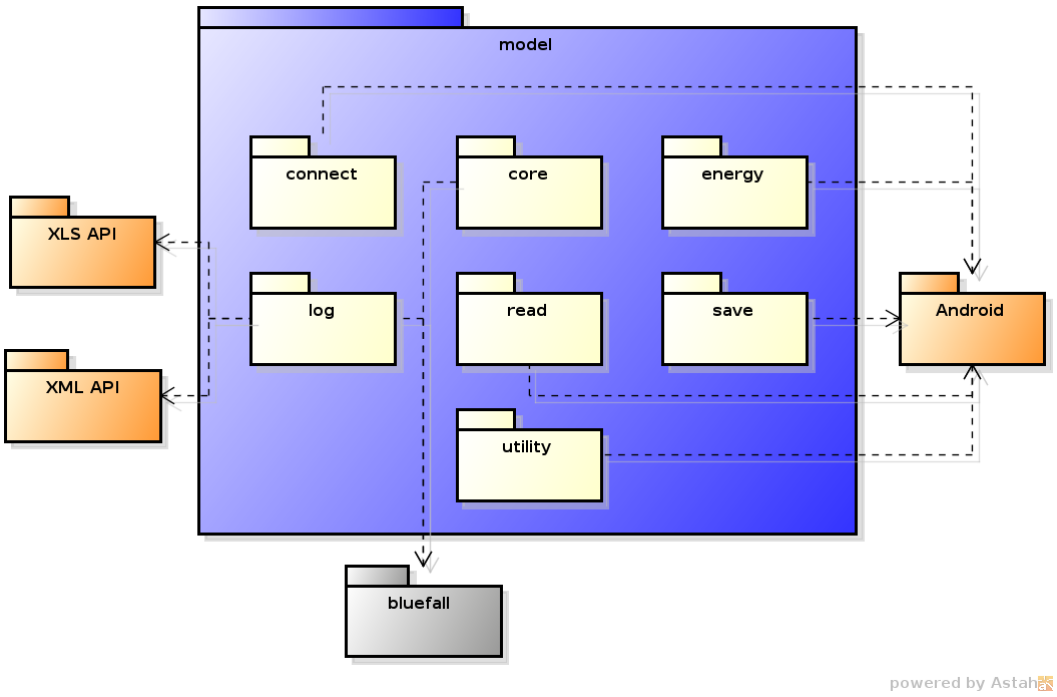


Figure 4.4: The “model” package scheme (mobile)

Figure 4.4 contains the structure of the “model” package and it highlights the dependencies that the packages have among them. In particular, since the Android framework does not provide any way to write XLS or XML some external libraries have to been used. Moreover both the “core” and the “log” packages contain the parameters defined by the user and some of their classes implement the “Observed” interface offered by the Observer design pattern in the “bluefall” global package. Finally, all

the other packages rely on some structures offered by the Android framework, that is symbolized by a single external package.

The "controller" package contains:

- all the activities that model the main stream of the application;
- a package to handle the reception, the transmission and the save of a file at the same time, called "threads";
- a package to browse into the file system of the device, called "fileExplorer", since the Android framework does not provide a ready-to-use way to do it.

The "threads" package contains only structures that encapsulate sequences of operations to be done in order to allow the three actions (reception, transmission and save) together, but it has to use other structures that allow to make available and to withdraw data between the actions: these structures are included in the "core" package of the model.

Although the "fileExplorer" package's aims may suggest to put this package among the "model" sub-packages, it deals with the visualization of a window that shows the contents of the filesystem indeed and it has also to handle the file which the user selects: for these motivations, the package is better collocated into the "controller" package.

4.2.2 Components on desktop platform

In C++ the corresponding of Java packages are "namespaces": in Java a package is physically represented with a folder, while this is not compulsory working with C++ namespaces. However, since the partition in folders is very convenient, each namespace matches a homonymous folder also in this version of the software. A global namespace has been designed, called "bluefall", whose direct sons are the corresponding three MVC namespaces, "model", "view" and "controller". Since the "Signal-Slot" mechanism offers a good way to implement some design patterns, like the Observer one, there are no classes or interfaces located in the global namespace.

The "model" namespace (Figure 4.5) is almost equal to the corresponding package in the Android version of the application. The only difference is that an "exception" namespace has been added: while in Android project almost any problematic situation is handled locally, in the desktop version the model structures have been designed to signal each problematic situation that could happen, leaving to the controller the burden to fix it, so the way used by model structures to signal a problem is throwing a particular set of exceptions, which are exactly encapsulated in the "exception" namespace.

The diagram in Figure 4.6 shows the dependencies of the sub-namespaces in the "model" namespace: the "connect" namespace has to use the external libraries for using Bluetooth and WiFi-Direct protocols, and both the "connect" and the "core" namespaces need the "exception" package to signal any problem during the usage of the structures made available in the namespaces. Moreover, the "read" namespace is linked both with a library that allows operations on remote files ("HTTP API") and with another one that allows operations on local files ("Disk I/O API"), while the "save" namespace is linked only with the last one since remote save of files is not supported yet by the application.

The "controller" namespace differs from the corresponding package since a namespace for the file browser is not needed: in fact the framework Qt provides structures

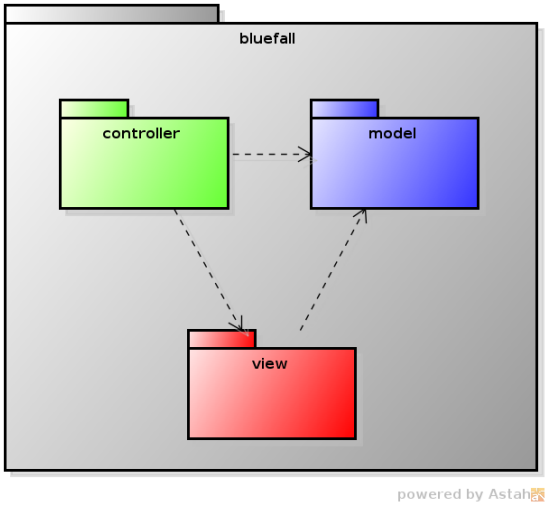


Figure 4.5: The "bluefall" package scheme (desktop)

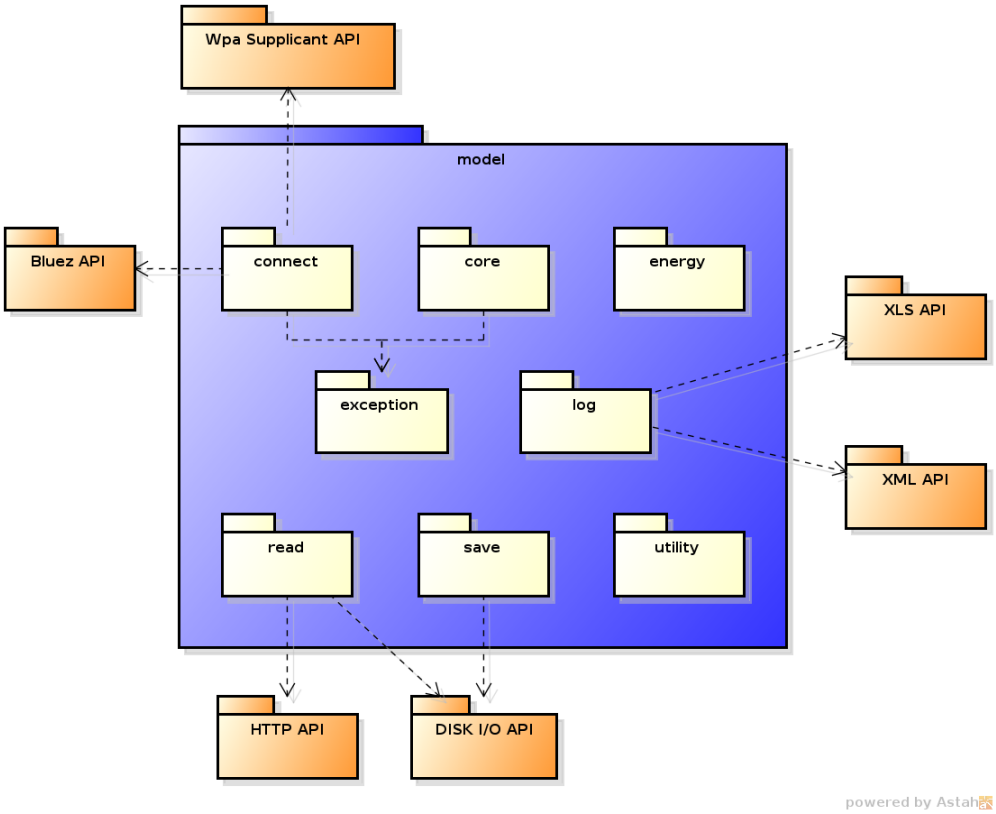


Figure 4.6: The "model" package scheme (desktop)

that allow the browsing of the storage memory in an easy way, so they have been used in substitution. Since there are no activities in the desktop version, the "controller" namespace contains all the structures that model the possible streams of operations.

The "view" namespace contains the modules whose aim is to handle the layouts that have been defined throw XML files. These modules take case also of performing a preliminary examination of the user input in order to accept it or to show an error message: in this way the controller receives the user input without having to check its consistency since it is granted by the view work. This trade-off allows to offload part of the controller's work on the view without weighting the view's components too much. The "view" namespace contains also the "dialog" sub-namespace, that includes the small window shown to the user once particular events happen, like waits or malfunctions.

4.3 Integration tests

The integration tests ensure the right cooperation between the macro-components identified in the architectural design. They result very useful when these components are parallel developed by different members of a developers team: in fact, in these cases the interfaces of the components are previously defined, so the developers create the components taking into account the specifications of the interface and then the components are put together in the build process. The integration tests support the build process ensuring that the components respect the interfaces designed and each time a change in a component is performed the tests that concern the component should be repeated, accepting or refusing the change committed depending on the results of the tests.

BlueFall has been developed using a bottom-up strategy, so the components that are small and with few dependencies have been developed first and then the more complex ones have been built up on the base created by the simpler ones. Once completed, each component has been built up against the rest of the software and heavily tested before proceeding in the development of the next one.

Chapter 5

Detailed design

The detailed design consists in the definition of classes and interfaces starting from the scaffolding that is provided at the output of the architectural design. The detailed design first takes care of defining which external libraries will be used in the project since classes and objects have to deal with the interfaces provided by them.

5.1 External libraries

An external library is needed when the main framework used in development does not provide the structures for a particular purpose. It is better to keep the number of external libraries used as low as possible since compatibility problems may arise among them. The first step is the definition of the needs of the application: doing so it is easy to understand which needs could be satisfied by the framework used itself.

The actions that BlueFall has to be able to perform can be summarized in:

- communication with other devices through Bluetooth;
- communication with other devices through WiFi-Direct;
- writing of XLS files;
- writing of XML files;
- download of files from Internet;
- I/O operations on storage memory.

The Android version of the application can perform almost all of the operations only using the Android framework: in fact, the framework is complete, easy to use and well documented, thanks also to a big number of Java standard libraries included. The only action that needs an external library is the writing of XLS files and the library chosen to satisfy the need is the JExcelApi library [12], which has a lot of examples that make easy understanding its way of working.

For what the desktop version of the application concerns, Qt framework allows only to perform Internet operations and I/O on storage memory. To communicate with Bluetooth the BlueZ library [13] has been used since it is the most famous and some guides to learn to use it can be found, while in order to use the WiFi-Direct protocol the WPA Supplicant library [14] has been almost a force choice since there

are not much tools that support the protocol. Since WPA Supplicant can handle only the MAC-level connection among devices, a IP-level connection handler is required: the packages udhcpc and udhcpd have been used, they are small tools that helps to configure DHCP connections. The writing of XLS files is performed through the usage of LibXL [15] while the writing of XML files is delegated to the fast RapidXml library [16]. Finally the Qt framework may fail in some Internet operations, like retrieving the size of the file to download: the cURL library [17] is used as a support tool when Qt shows some problems.

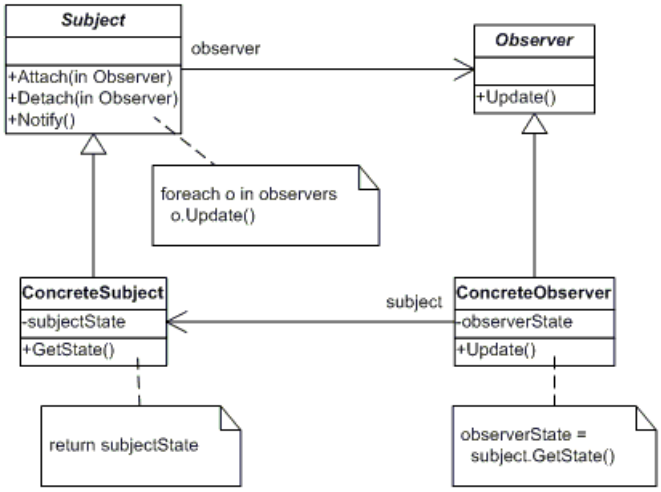


Figure 5.1: The Observer design pattern

5.2 Design patterns

The usage of design patterns helps to create more maintainable and less coupled code. The most used pattern in both the Android and the desktop versions of the application is Singleton: it is particularly suitable since a lot of model objects are used for a limited amount of time but they are needed later however. Moreover the application needs some objects that take care of storing the parameters in input from the user and making them available to every other object that needs them, so Singleton pattern is perfect for this type of purpose. The Singleton pattern is often used together with a builder object, that corresponds to a structure which provides the instances of some classes returning a generic reference to them: in few words, the builder is the only one which takes care to build objects of some classes concretely and returns a generic reference of the just created object to the requester. In this way the requester does not touch the real implementation of the object needed but it handles only its interface. Moreover, working with Singleton objects, once the objects have been created, the builder has only to return the generic reference, so the "real" build is performed only once per class.

Another used design pattern is the Observer one (Figure 5.1): it is particularly useful with objects that always need updated information from the objects that store the user parameters. In fact, in the Android version of the software the objects that gather all the parameters implement the "Observed" interface (they inherit from an abstract class indeed) so every object that needs real-time updates on the parameters

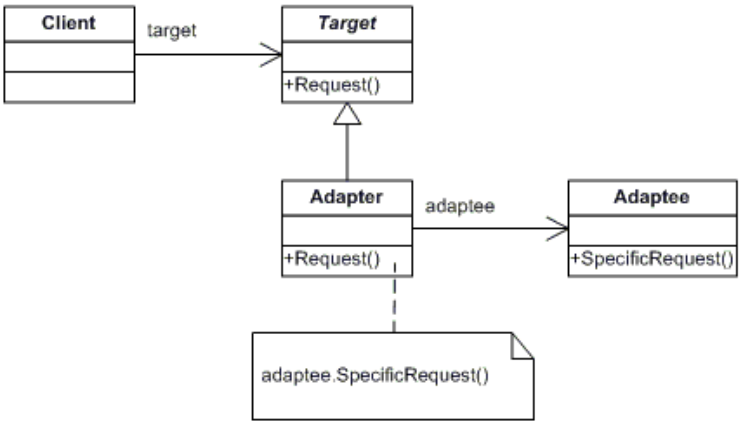


Figure 5.2: The Adapter design pattern

can add itself to the lists of those which are notified once an update is available. In the desktop version the pattern is made through the Signal-Slot mechanism instead and this way of working is even better than the traditional pattern since the "updater" object does not need to hold a reference to every object that needs updates but it just send a broadcast signal that is listened by those that need updates.

The Adapter design pattern (Figure 5.2) has been heavily used as point of conjunction between the software and the external libraries. A common interface has been defined (for example, the "Connector" interface or the "LogWriter" interface) then a concrete object implements the interface (for example, the "BluetoothConnector" or the "XLSWriter"): this one uses the interfaces and the objects provided by the external library. In this way a controller module can use the external library only handling a generic reference. The Adapter pattern has been applied on the structures that communicate through Bluetooth and WiFi-Direct, on the classes that take care about writing the log files and on the modules used to read and write data.

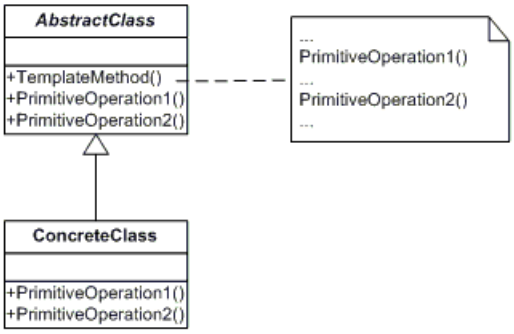


Figure 5.3: The Template Method design pattern

Finally the Template Method pattern (Figure 5.3) has been used in the desktop version of the application. Each view component allows the change of the screen of the application with another view component only if an undefined number of checks give positive result: in fact these checks highly depends on the component itself, on

the elements that are displayed and on the user inputs. In order to allow each view component to define its own checks, a common interface (abstract class) has been defined for all the view components and the concrete method that allows the passage from a component to another one ("isComplete" method) has been redefined: now this method encapsulates a call to the abstract method "check" whose aim is to return true only if any control encapsulated in it gives positive result. In this way each interface implementer has to define the "check" method and it can put here all the specific checks that have to be performed before passing to another component.

5.3 Management of threads

One of the most interesting aspects of the application is that it allows to receive and to transmit a file at the same time. This can be easily achieved through the usage of two different threads, one used to receive the resource and one used to send it towards another device. Moreover, a third thread should be used in order to save the file received on the storage memory. However, the threads need some synchronization mechanism since both the transmitter and the saver threads depends on the data brought by the receiver thread: how to handle data that have to be shared among threads? A good solution is to put data in a common buffer, so the receiver thread can put the data retrieved into the buffer and the other two threads can read the data available in the buffer. These elements consist in the basic components of the producer-consumer pattern: in fact, the receiver thread can be seen as the producer since it puts data into the buffer, while the transmitter and the saver threads can be seen as two consumers since they consume (read) the items (bytes) put in the buffer by the producer.

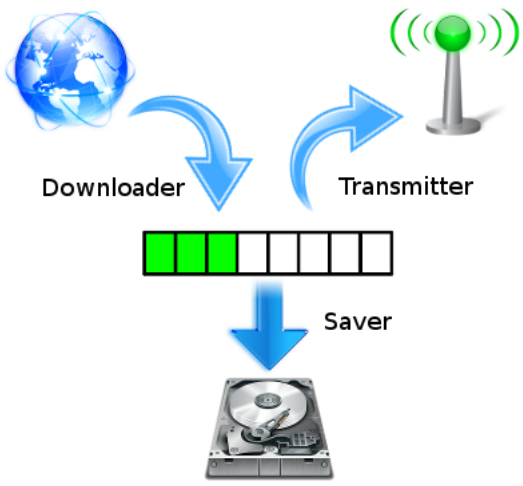


Figure 5.4: Threads working on the same buffer

Figure 5.4 shows how the three threads work in a device which retrieves a file from Internet: the receiver thread (called "Downloader") is represented by the arrow that goes from Internet to the buffer and it is the only thread that brings data into the buffer, the transmitter thread is represented by the arrow that goes from the buffer

to the antenna while the saver is the arrow which goes from the buffer to the disk. There are some variants that may occur: for example, if it has been decided to share a local file instead of a remote one, the "Downloader" thread will become a simple file reader thread, or if the device is assigned the intermediate node role than the "Downloader" thread will retrieve data from the specific device used by the protocol adopted in sharing, like an antenna.

5.4 Modules on Android platform

The classes contained in the packages that belong to the model macro-components have a similar structure among them. Figure 5.5 shows the structure of the "log" package: the packages "connect", "read" and "write" reflect an almost identical pattern. In the package a common interface can be identified (the "LogWriter" abstract class): this one has to be implemented by all the type of log writers that the application supports. In fact, both the "ExcelLogWriter" and the "XMLLogWriter" classes inherit from the "LogWriter" abstract class and the common interface represents the type with which controller components have to handle every type of log writer. Moreover the classes that need to use a log writer do not build the object by themselves but they use the "LogWriterBuilder" class instead: this class is the only one that can build log writers concretely, returning an interface reference to them. In this way the user of the log writer do not know the implementation of the log writer needed and the only class that touches directly the concrete implementation is the builder one, keeping low the coupling among the classes.

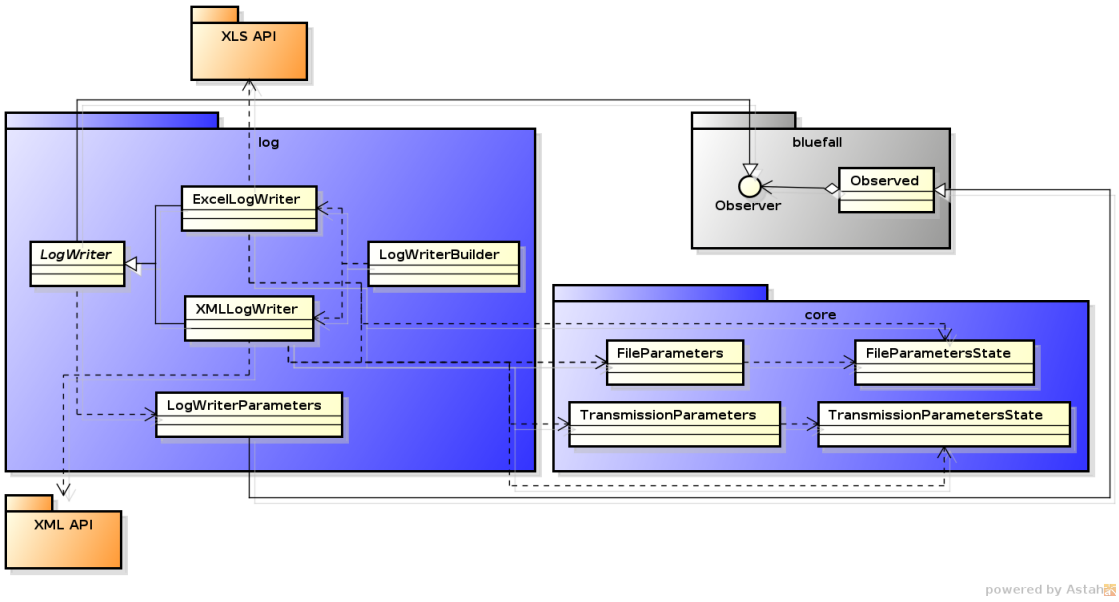


Figure 5.5: The "log" package (mobile)

Another interesting thing is the binding between the "LogWriter" abstract class and the "LogWriterParameters" class: the last one is the place where all the parameters that concerns the log writing are stored, like the log file format and which has to be stored in the log file or not. Since every log writer has obviously to know which

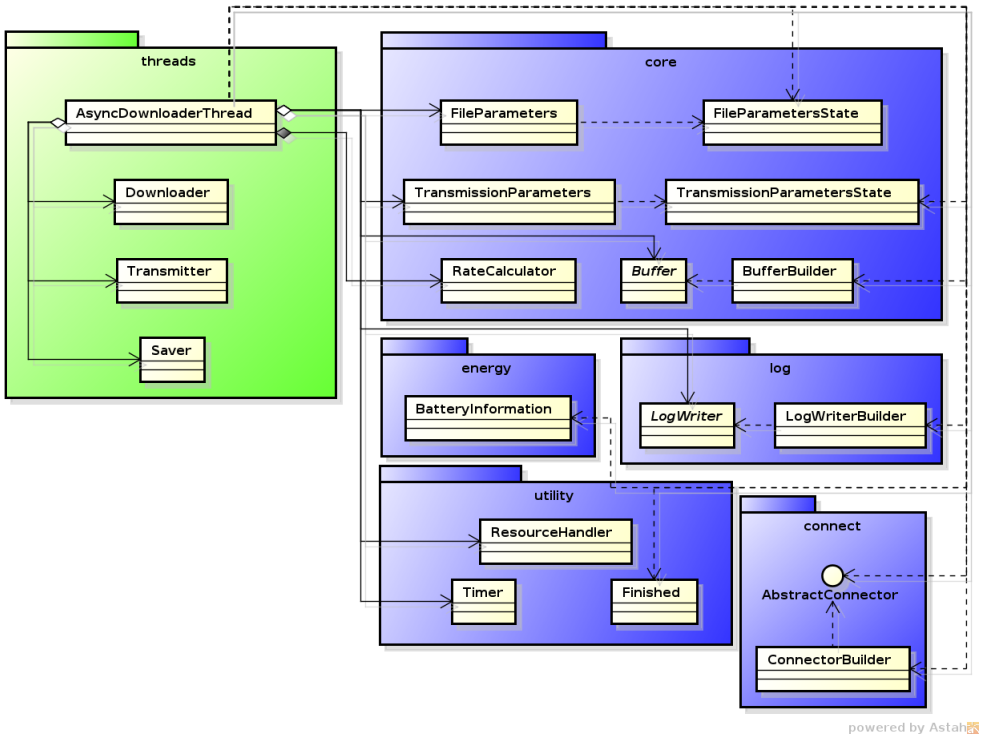


Figure 5.6: The "thread" package classes from the "AsyncDownloaderThread" point of view

are the parameters defined by the user, a direct link between the "LogWriter" class and the "LogWriterParameters" class could be a solution, but since the parameters may be changed during the execution of the software, the log writers need to have the updated version of the parameters. These elements led to the choice of the Observer pattern as the best way to connect each log writer to the parameters storer: in fact, each time the parameters change their updates are sent to each log writer, as the Observer pattern appoints.

The controller macro-component has been used as deposit for all the activities defined: the only problem that this decision may involve is that the package could become crowded once the number of activities grows, but this number is quite low at present and any repartition of the activities into separated sub-packages does not lead to significant advantages. Into the controller component the "thread" package has been defined in order to allow the reception, the transmission and the save of a file at the same time. The package contains three classes that correspond to the three activities, so the "Download" class matches the reception, the "Transmitter" class matches the transmission and the "Saver" class matches the save of the file, and a further class, the "AsyncDownloaderThread", whose purposes are:

- to prepare the environment to allow the other 3 threads to work correctly;
- to start the other three threads;
- to measure the time involved in sharing;

- to measure the energy consumption;
- to measure the data rate, showing the results on the screen;
- to make samples on the sharing, storing the data in the log file;
- to stop the threads in case of user interruption.

These actions are performed in a separated thread since the main thread of the application, the so-called "UI thread", must be left as free as possible, because its aim is to listen for user inputs, so offloading long works on the UI thread leads to the crash of the application. The main disadvantage of this solution is that the high number of actions that the thread has to perform causes it to need a lot of references to model objects, as Figure 5.6 shows: this makes growing the degree of coupling among the controller and the model.

So when the sharing is being performed, up to 5 threads run concurrently in the application: if the device has been assigned the last node role or the receiver role it is not needed to make the "Transmitter" thread start since there are no other devices which expect data from it; in the same way, if the device has assigned the first node role or the transmitter role and a local file is being shared there is no need to start the "Saver" thread since the file does not need to be re-saved.

5.5 Modules on desktop platform

The structures in the model macro-component of the desktop version of the application are very similar to those of the mobile version. Figure 5.7 shows the inner structure of the "connect" namespace, and comparing it with the "log" package of the mobile version some common elements can be identified, like:

- a common interface represented by an abstract class, called "Connector" in this case;
- some implementers classes, that are "BluetoothConnector" and "WiFiDirectConnector" in this case;
- a builder class, called "ConnectorBuilder" in this case.

This structure is particularly good from the point of view of any further development since it is very easy to add a new "Connector" implementation: the extension of BlueFall with a "NFCConnector", for example, just implies to subclass from the "Connector" interface and to add the possibility to build the new connector to the builder object. Moreover the "build" method of each builder wants a string as identifier of the type of object to be built and this lead to the fact that a potentially limitless number of different implementations can be added to the software.

The "connect" package is interesting since it also shows two examples of application of the Adapter design pattern: each one of the implementations of the "Connector" interface deals with an external library but thanks to the application of the pattern this is completely transparent to the final user of the connector because he will handle only a generic reference to a "Connector". In this way if the external library has to be substituted because of obsolescence or malfunctions the only class that has to be modified is the "Connector" implementer, the real adapter.

The relationships between the controller components and the view ones are the most interesting aspect of the design: the already discussed QWizard structures is

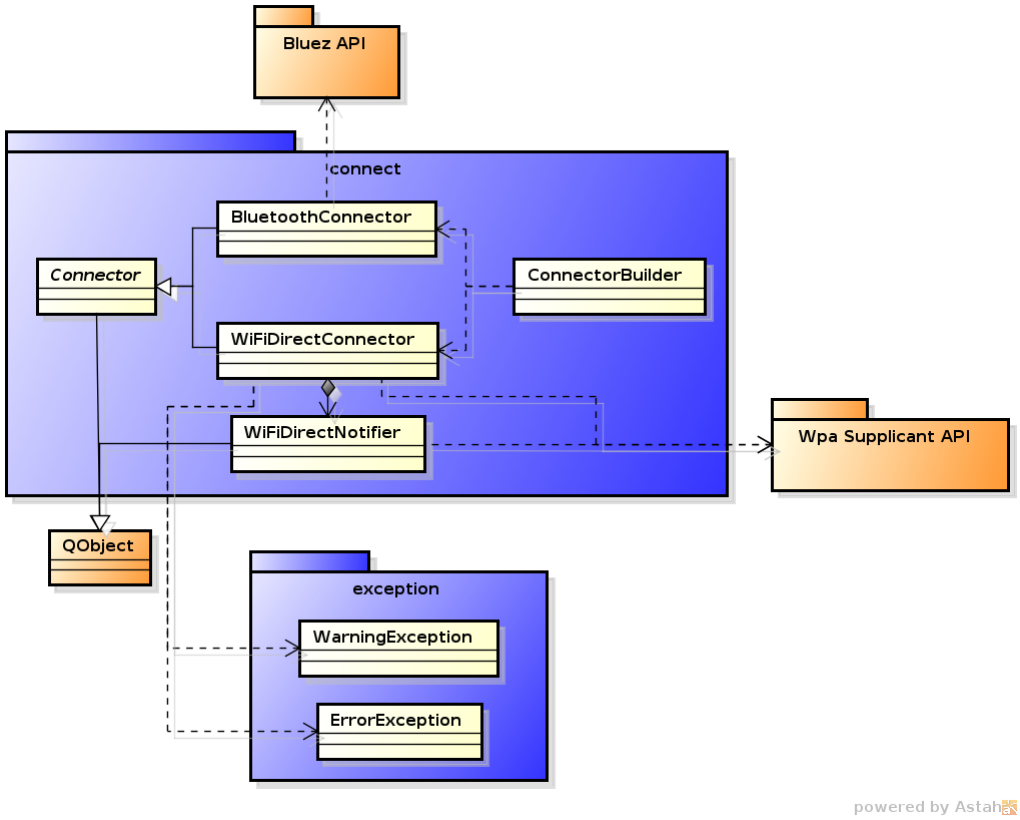


Figure 5.7: The "connect" namespace (desktop)

composed by a set of QWizardPage(s) which are objects that handle a single page in the wizard. Each QWizardPage can be considered a view component since its aims are to display the elements of the graphical interface and to listen to the user inputs: in the same way, a controller component for each page is needed to perform the operations that have to be done once the final user interacts with the graphical interface. How to connect the view and the controller components of the same page? A solution could be that the controller component handles a reference to the view one but a direct binding between two implementations is really bad from the maintenance point of view. So a communication through interfaces might be a good solution: both the controller component and the view one could inherit from a generic interface and the controller component could communicate with the handled view component through it. Even if this solution may seem good the two interfaces are too much generic to allow each controller-view couple to work properly: in fact the purposes of the pages are so different that it is almost impossible to find a common interface among all the controller components. The same consideration is valid also for the view components.

The solution adopted consists in the definition of 5 empty methods in the interfaces of the controller and of the view components: these methods can be overridden by the implementers of the interfaces depending on the needs. These "custom" methods can be used to encapsulate the specific code that has to be executed in order to make the controller-view couple working and above all the couple can keep on communicating

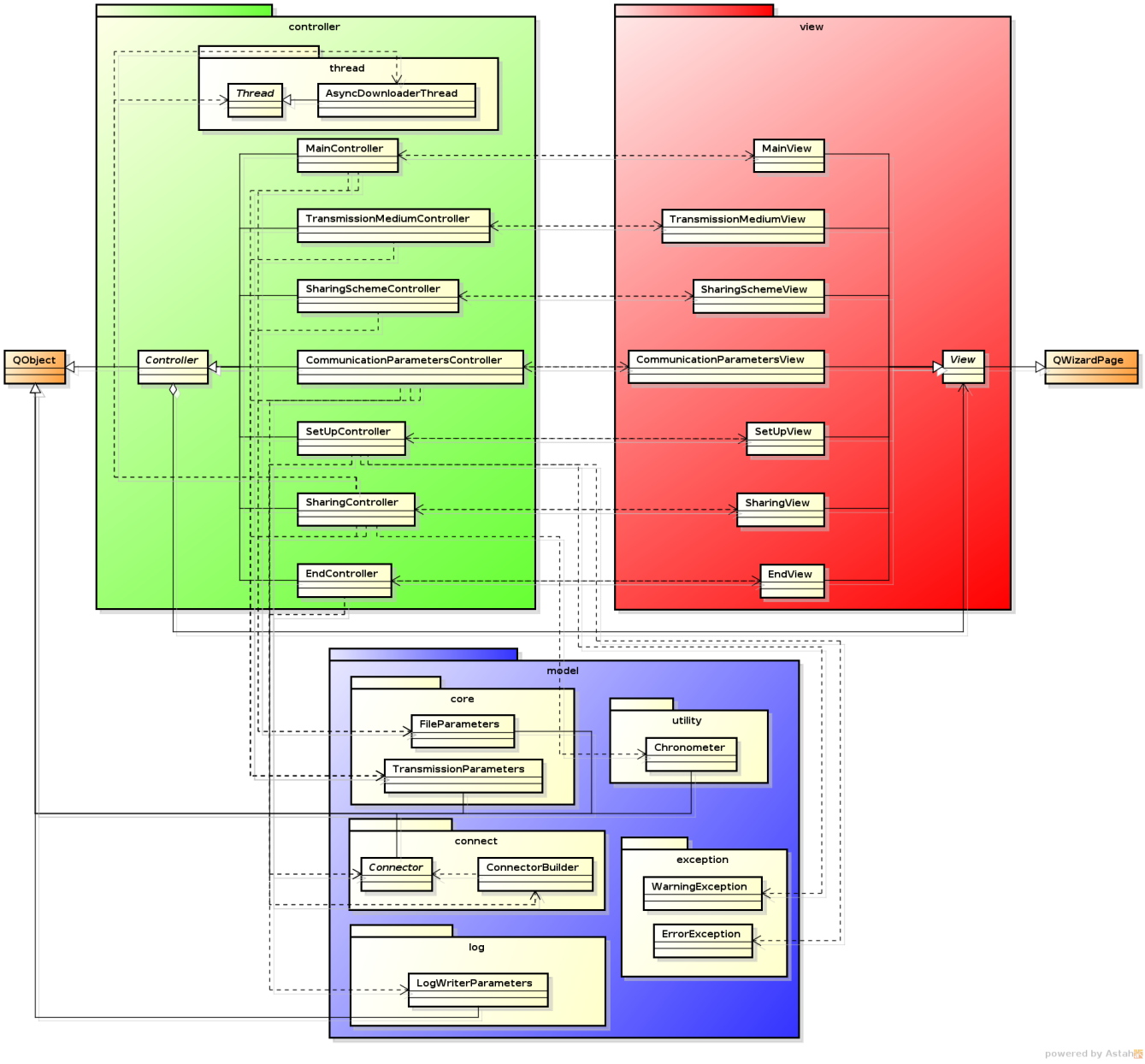


Figure 5.8: The "controller" namespace (desktop)

through interfaces.

An example: a "ConcreteController" object has to handle a "ConcreteView" object and both of them have been designed for a specific page of the wizard. The "ConcreteController" inherits from a generic "Controller" interface and also the "ConcreteView" inherits from a generic "View" interface: the "ConcreteController" can handle the "ConcreteView" only through a generic reference to it (a "View" reference) provided by the "Controller" interface. To let the "ConcreteController" doing some specific operations on the "ConcreteView", the "View" interface provides 5 empty methods (called "customMethod1", "customMethod2", and so on) that the "ConcreteView" can override in order to encapsulate some specific code. So the "ConcreteController" can execute the generic method "customMethod1" on the "View" reference being aware of the real effects of the method invocation. The disadvantage of this solution is that the names of the methods are not self-explicative on the aims of the method implementation since their purposes are undefined: to use this type of solution the design of the controller-view couple of components must include a sort of "contract" which explains the purposes of the overriding of the methods, so that each one that wants to modify the couple can understand the meaning of the methods calls.

How to pass parameters to these methods? No one of the primitive types fits since the purposes of the methods are undefined. Qt comes in help with the QVariant class, which defines objects that can be easily converted to each primitive type and if the override of a "custom" method needs more than one parameter, a QVariant can also be converted to a list of arguments. The only overhead is the conversion from QVariant to the expected type of parameter but this burden is trivial if compared with the benefits of the QVariant usage.

Moreover, the Signal-Slot mechanism offered by Qt framework helps to improve this solution: in fact, to avoid direct calls to methods, both the controller interface and the view interface defines 5 "custom signals" and 5 "custom slots", so each implementer of the interfaces can override the slots (in place of the methods of the previous example) in order to perform specific operations. In this way the only place in which the controller component uses the view reference is in the "connect" instructions, in which it defines the responses of the view (slot executions) to its signals and its responses (slot executions) to the signals of the view. In Figure 5.8 the Signal-Slot communication between a controller and a view component is represented by the double-headed dotted arrow and it is possible to note the generic reference to the "View" interface that the "Controller" interface owns. It is important to note that each view component that handles a page of the wizard does not know anything about any controller component, so each connection between signals and slots has to be performed by the controller: this behaviour put the view components at the same level of the model components, so they are "tools" that the controller components use when needed, allowing a parallel development of the view and of the controller components once the interfaces (the "contract(s)") have been defined.

5.6 Unit tests

The unit tests allow to ensure the correctness of small parts of the software, like single classes. Their aim is to check that the response expected from the code execution corresponds exactly to the one provided, so their real usefulness is to find unexpected errors in the code. To perform a high number of tests on the code it is generally helpful to rely on a framework: it provides tools that make the test development easier and

that allow to locate any error during the tests.

One of the most interesting aspects of testing is mocking: when an object mocks another one, it takes the appearance of the mocked object and it can be used in substitution. The mock object can be configured to provide decided answers when methods are called on it: this aspect is particularly important because in a lot of scenarios testing an object which depends on other objects should involve also testing the related objects, but in the most of the cases this is not a desired consequence. With mocking it is possible to keep alone the object to be tested simulating the other related objects with pre-set answers.

The Android framework provides some libraries designed for the purpose of testing that have been heavily used. The most important feature of these libraries is that they allow to simulate a classic user interaction, like button pressures and text writing, so they result particularly indicated in tests that concern the graphical user interface. However these libraries are very poor in mocking ability: in fact the Android testing libraries provide only some pre-built mock objects whose answers to method invocations are not configurable, so it is not possible to mock a user defined object. These limitations led to search for new testing tools allowing to create customizable mocks. Two useful libraries have been found: Mockito [19] and PowerMock [20].

Figure 5.9: The two main tools used in testing the Android version of BlueFall

Mockito provides a very comfortable way to create mocks for almost every type of user defined object: the library is very common among the Java testers and it is easy to find complete examples which allow to understand its way of working. However Mockito is not able to mock some particular objects: these are final (not overridable) objects for example. In these scenarios PowerMock can be used: it has a syntax almost identical to Mockito's one and it is able to mock the objects on which Mockito gives up using the bytecode manipulation. Nevertheless there are some compatibility problems between PowerMock and Android platform since PowerMock has been designed for pure Java applications which works on the JVM, while Android platform works with the Dalvik virtual machine, that is a particular version of the JVM optimized for devices with low amount of RAM, like phones and tablets. In order to avoid these compatibility problems it is possible to run an Android application on the JVM only for testing purposes: this workaround allows to use PowerMock on Android projects, even if the testing possibilities are really limited since no one of the Android framework objects can be used on the JVM. Finally BlueFall tests have been divided into two sets: one set of tests is executed on the Dalvik virtual machine and it includes graphic interface tests, activities works checks and "light" mocking, while the other set of tests is executed on the JVM and it includes the tests with "hard" mocking relating code more difficult to test.

The desktop version of the application has been tested through two different libraries: QTestLib [21] and CppUTest [22]. QTestLib is the tool that the Qt framework provides to test the Qt-based applications: it is easy to use and it is also well doc-

umented. It is particularly appropriate to test the view components since it is able to simulate clicks, text writings, mouse movements and so on, just like the Android testing framework does.

Figure 5.10: The two main tools used in testing the desktop version of BlueFall

The main disadvantage of QTestLib is that it does not support mocking: for this reason it has not been used too much. The testing library had both to support mocking and to be able to mock also C functions, since some of the external libraries used by the application are written in C: a first choice was CMocka, a very light library for mocking C functions, but it had some compatibility problem with the application so it was abandoned. Then CppUTest has been tried and so it has become the mainly used testing library. In fact, CppUTest is quite easy to use thanks to the manual provided in the official site, and it allows to mock both classic C functions and methods invocations on objects.

The parts of the code that have been mainly tested are the ones which concern the data rate measurements, the time measurements and the buffer consistency since they represent the most important elements of the application. Generally the model components represents the easiest part to test since its modules have few dependencies and a low number of bindings with other objects, while the controller components are difficult to test because a lot of objects have to be mocked and this is not always possible.

Chapter 6

Realization

In the realization process two parts can be recognized: the coding of the software and the drawing up of the documentation. The following sections describe the two activities in detail.

6.1 Coding

Some aspects of the coding process are noteworthy: first of all, it has been decided to use some features of C++11, the latest C++ standard, in coding of the desktop application. These are smart pointers and threads.

In C++ the handling of the application memory (heap) is entrusted to the programmers: if from one side the deletion instructions at compile time are extremely efficient, from the other side it may be really hard to track all the objects on the heap and to delete them at the right moment. In Java this problem has been solved through the usage of a garbage collector: it is a thread of the JVM which deletes objects from the heap whose number of active references has dropped to zero. While this solution is very comfortable from the programmer's point of view, it cannot compete with the efficiency of the user defined deletion of objects. So how to keep the C++ efficiency in a more convenient way? Smart pointers have been designed as "pointers which get by deleting the occupied memory by the pointed object in the right moment": they implement an internal reference counting, so when the counting drops to zero they delete the pointed object. The new C++ standard library offers some structures that trace this pattern: these are the "shared pointers" and the "unique pointers". The first has to be used when the pointed object could be referenced by many pointers, while the second one is for objects with an only point of access. These structures have been heavily used in all the project, keeping the number of traditional pointers as low as possible: this ensures a certain degree of correctness in memory handling.

The C++11 standard library defines also threads and a set of structures that allow synchronization among them. Standard threads have been used in BlueFall in an indirect way: the project relies on the QThread class, the threads provided by the Qt framework, that are based on C++11 threads in turn. To coordinate the threads activities the C++11 standard library does not offer a monitor-based mechanism like the JVM, on the contrary, it provides mutex: even if they are not convenient as the Java offered mechanism, they are quite simple to use however, so they have been used to apply the producer-consumer pattern explained in Section 5.3.

For the development of the Android application the main tool used to write code has been Eclipse [23]: it provides automatic identification of syntax errors in the code, compilation during code writing and the Android framework integrates very well with it. For the development of the desktop version, the Qt framework provides its own tool as development environment, named QtCreator [24]: it offers a WYSIWYG editor for GUI like the one provided by the Android framework and it allows to define the signal-slot bindings among visual elements though a graphical interface.

The most interesting side of coding is the problem solving: the following section describes the problems encountered and the solutions adopted.

6.1.1 Problems and solutions

The first dilemma came out from the producer-consumer pattern application and it concerned the dimensions of the buffer. In the traditional producer-consumer pattern the two threads have to produce and to consume an undefined number of items, so 2 conditions may happen:

- buffer empty, so the consumer thread has to stop;
- buffer full, so the producer thread has to stop.

In real applications of the pattern the number of items is generally known: in particular, in BlueFall case the number of items corresponds to the number of bytes of the file being shared. So which length has to be assigned to the common buffer? There are 2 possibilities.

- Creating a buffer smaller than the size of the file allow to trace the traditional pattern better since the "buffer full" condition may happen. Moreover it allows to share files of any size since the memory used is independent from the size of the file being shared. The bad aspect of this solution is that the reception of a file is not decoupled from its transmission and from its save: if a particularly slow protocol is used in sharing, the retrieving of the file from Internet would proceed in fits and starts since a lot of times the "buffer full" condition will occur.
- Creating a buffer of the size of the file to be shared eliminates the possibility that the "buffer full" condition may occur since the receiver thread has never to stop till the entire resource has been retrieved, then its work is finished. The solution allows a real decoupling of the activities since the retrieving of the resource is not constrained by the speed of the transmitter activity. The biggest problem of the solution is that the dimension of the file to be shared is limited to the amount of RAM that the application is granted to use: this is a big limit on phones and tablets, whose central memory amount is not so high, while in the desktop version of the application this may result not very important.

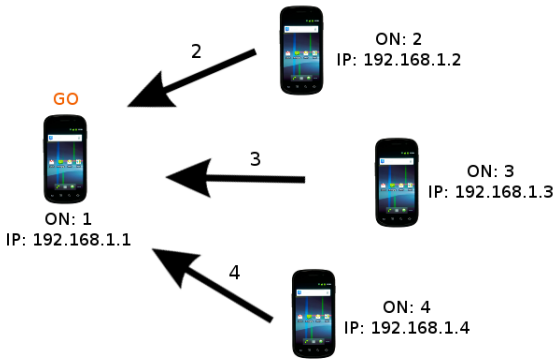
The second solution has been adopted in BlueFall since one of the most important features the application must have is to perform measurements: in this context it is very important to leave the threads less constrained as possible. Moreover the sharing of a big file can be transformed in the sharing of a list of small parts of the big file, so adding a quite simple decomposition and reconstruction at the begin and at the end of the sharing respectively can be a workaround of the problem.

Another problem that has been faced during the development of the application considers the assignment and the distribution of the IP addresses when WiFi-Direct

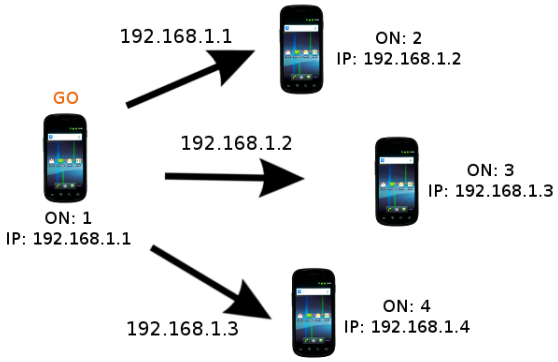
Figure 6.1: WiFi-Direct group scheme

is used as wireless protocol. In fact, while with Bluetooth a MAC-level connection is enough to allow the devices to exchange data through sockets, with WiFi-Direct a MAC-level connection is firstly needed, than an IP-level connection has to be performed in order to gain the sockets that allow to send and receive data. As in almost every type of connection, the device which performs the connection has to know the address of the device to connect to, while the other one has to be in listening for incoming connection to accept any connection request: the same rules are valid also for the IP-level connections. Before proceeding in the description of the problem, a premise is needed: to create a WiFi-Direct network, a device has to create a WiFi-Direct group to which other devices can join. The group creator is called "group owner" (GO) while the joiners are called "clients" (Figure 6.1). When a device joins a WiFi-Direct group, this one can obtain the IP address of the group owner: in the same way, the group owner can obtain the IP address of each device which has joined its group. In this scenario applying the star sharing scheme is not a problem: it is sufficient to coincide the group owner with the device to which the transmitter role is assigned and the clients with the receiver devices. The receivers are able to obtain the transmitter IP address, so they can obtain a socket to it and also the transmitter can obtain a socket for each device of the group. The problems arise applying the waterfall sharing scheme to a set of WiFi-Direct devices: how is it possible for each device to know the IP address of another different device in the set? A solution may be to create a WiFi-Direct group for each couple of devices: the first node device creates a group with the second device in the waterfall, the second device creates a group with the third one, and so on. Unfortunately on Android devices the participation to more than one WiFi-Direct group is an unsupported feature, so that the solution does not fit. The only alternative is to create an unique WiFi-Direct group among all the participants of the waterfall: in this way each device (except the group owner) can know only the IP address of the group owner, as said before. How can a device know the IP address of another device which differs from the group owner?

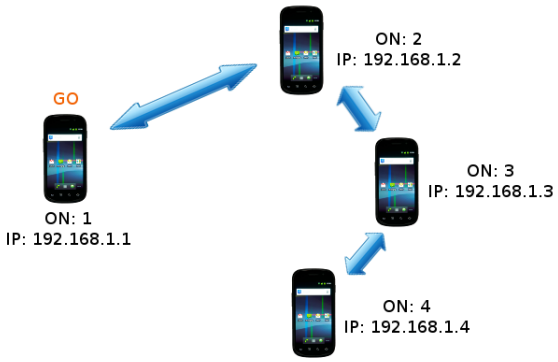
To solve the problem it was exploited the fact that the group owner is able to gain the IP addresses of all the clients which have joined the group, it just needs to know which device needs which IP address, so each participant of the waterfall is asked to provide another parameter, called "order number", which represents the position of the device in the waterfall, between 1 (the order number of the first node) and n,



(a) First stage



(b) Second stage



(c) Third stage

Figure 6.2: The three stages of the IP addresses distribution

where n is the total number of the devices involved in the waterfall (which corresponds to the order number of the last node). So the device immediately following the first node will have 2 as order number and the last but one will have *n-1*, for example. The first thing to do is to coincide the first node with the group owner: in this way the device has assigned 1 as order number automatically. Then, once all the clients have joined the group formed by the group owner, a "waterfall setup" phase is started, which consists of three stages (Figure 6.2).

1. Each one of the clients sends its order number to the group owner. The group owner puts the number received in a hash table in which the keys are the IP addresses of the clients and the values are the order numbers received.

2. Once all the order number have been received, the group owner sends to each device the IP address of the device immediately preceding. So the IP address of the device whose order number is 1 will be sent to the device whose order number is 2, the IP address of the device whose order number is 2 will be sent to the device whose order number is 3, and so on.

3. Once all the devices have received the IP address they needed, each one of them can perform the connection to the right device.

After all the devices have gained to obtain a socket to the target device the waterfall is complete: now each device receives data from a socket and transmits the same data to another socket, except for the first and the last node obviously.

During the first tests of the "waterfall setup" phase some errors occurred during the sockets creations between some devices, especially the ones located in the final part of the waterfall. Looking to the error logs it was discovered that a device (the point from which the waterfall building failed) was not able to obtain the socket to its immediately preceding device. The socket creation among two devices requires that a device puts itself in listening for requests and the other one performs the request: in this way both of them obtain a socket to the other device. The error that happened during the "waterfall setup" phase was due to the fact that the device performed a socket request to the immediately preceding device, which was not listening for incoming requests: the motivation of this behaviour was due to the fact that the device was still performing a socket request to its immediately preceding device in turn. In few words, the device put itself in listening for socket requests too late compared to the moment in which the following device performed the socket request, causing the described error. The motivation of this error is that each device in the waterfall first tried to obtain the socket to the immediately preceding device and then it put itself in listening for socket requests as soon as the "waterfall setup" phase started: in this way the devices in the final part of the waterfall performed socket requests when the devices of the first part of the waterfall are still creating sockets among them. To solve the problem, a wait is needed: in fact, if the device which performs the socket request waits before performing the request, the preceding device manages to obtain the socket of its preceding device and to put itself in listening for socket requests. But how much has a device to wait before performing its socket request? In BlueFall each device waits an amount of time which is proportional to the order number assigned to the device: in this way the first devices in the waterfall waits for a short time while the last devices in the waterfall waits more than the first ones, allowing each device to gain its socket to the preceding device and to put itself in listening for incoming requests. A smartest solution is to decouple the two actions: if each device starts two threads, one to perform the socket request to the preceding device and one to put itself

in listening for socket requests, no waits are needed and also the devices in the last part of the waterfall can perform the requests without errors. Then a synchronization point is needed: a third thread (the starter of the other two) waits the two threads to have finished their work (performs a *join* on the two threads).

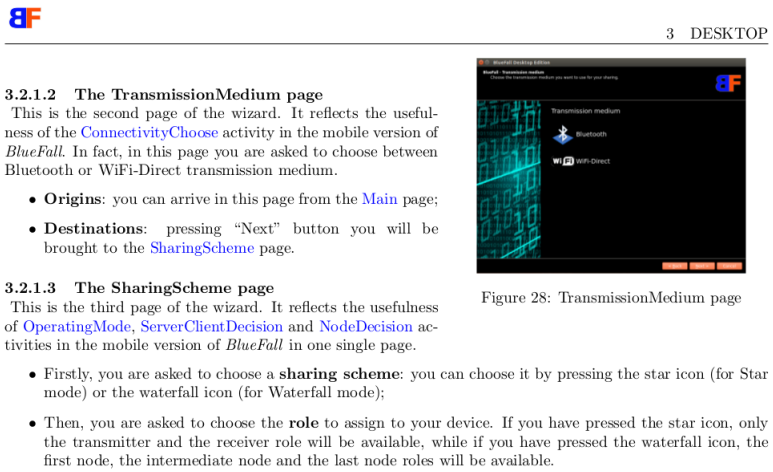
(a) Order in which the devices put themselves in listening for feedback

(b) Order in which the devices send feedback

Figure 6.3: The feedback mechanism applied in waterfall sharing scheme

Finally there has been a problem when multiple resources shares were performed: the moment in which the problem arose was between the finish of the sharing that concerned a resource and the start of the sharing that concerned the next resource. When there are multiple resources to be shared, once all the bytes of a resource have been received, the information of the next resource (name and size) are expected: in this scenario some of the devices failed to read the initial information of the next resource, gaining only incomprehensible data. This was due to the fact that the data transmission among the devices in BlueFall is quite rough since structures to encapsulate data, like packages or chunks, are not used, so bytes are sent and received without meta-information attached: the interpretation of the data received is offloaded on the receiver, which has to keep trace of the number of bytes received for each resource. In this context is quite easy to have errors since if a device reads just a single extra byte from the target socket and it discards it (the number of bytes of the resource being shared has been reached), the reception of the information relating the next resource is already compromised. This was exactly the cause of the problem: since the maximum number of bytes to read from sockets is pre-set, at the end of the sharing of a resource more bytes that the needed ones may be read and discarding them causes the problem described. To solve the problem, a feedback mechanism has been introduced: once a resource has been shared, before starting to send the information about the next resource, each device waits a feedback from the following device. In this way, since the application of the feedback mechanism coincides with the passage from the sharing of a resource to the sharing of the next resource, the device which sends the feedback is aware of the fact that the data that it will receive are the information about the next resource, so no wrong interpretations of the bytes received can be performed. The feedback generally consists in a single integer: in this way the mechanism application is quick and the number received can be compared to the expected one, in order to reduce the risk that some bytes received could be interpreted as the reception of a feedback. Applying this solution on a waterfall

results in a synchronization mechanism: in fact, each device waits that the next one sends it a feedback, till the last device, which does not wait any feedback but it sends a feedback instead that is received by the last but one device, which then sends a feedback in turn, and so on (Figure 6.3). The solution is also applied on star scheme to grant the consistency of the state of each receiver: the transmitter device waits a feedback from each one of the receiver before proceeding in the sharing of the next resource.



3 DESKTOP

3.2.1.2 The TransmissionMedium page

This is the second page of the wizard. It reflects the usefulness of the ConnectivityChoose activity in the mobile version of *BlueFall*. In fact, in this page you are asked to choose between Bluetooth or WiFi-Direct transmission medium.

- **Origins**: you can arrive in this page from the Main page;
- **Destinations**: pressing "Next" button you will be brought to the SharingScheme page.

Figure 28: TransmissionMedium page

3.2.1.3 The SharingScheme page

This is the third page of the wizard. It reflects the usefulness of OperatingMode, ServerClientDecision and NodeDecision activities in the mobile version of *BlueFall* in one single page.

- Firstly, you are asked to choose a **sharing scheme**: you can choose it by pressing the star icon (for Star mode) or the waterfall icon (for Waterfall mode);
- Then, you are asked to choose the **role** to assign to your device. If you have pressed the star icon, only the transmitter and the receiver role will be available, while if you have pressed the waterfall icon, the first node, the intermediate node and the last node roles will be available.

Figure 6.4: A manual page describing the steps of BlueFall configuration

6.2 Documentation

Some of the requirements in the specifications of the software concern the documentation of the software. In particular, the instructions to install, use and maintain the testbed are requested as mandatory requirement, while writing them in English language is considered a desirable requirement. In order to fulfil these requirements, a user manual has been written: it concerns the three operations exposed in the mandatory requirement (setup, usage and maintenance) both of the Android and the desktop versions of the application. To support better the maintenance, the documentation of the source code is also provided.

The user manual is primarily divided into two sections which correspond to the two version of the application. Into each section, three parts can be identified.

- The installation part, which explains how the environment has to be prepared in order to correctly install the application. It includes any list of needed libraries and step-by-step guides to overcome the classic obstacles in the procedure.
- The usage part, which shows how to use BlueFall in order to build up a network with one of the presented sharing schemes. The guides cover all the offered combinations of wireless protocols and sharing schemes, providing an image for almost each step of the procedure.
- The development part, in which the inner structure of the application is described. For the Android version the activities structure is shown, while for the

desktop version an explanation of the QWizard scheme and of its way of working is provided.

Since the development parts of the manual take care only to describe the high level structure of the application and how the application should be extended, a very detailed documentation on the current modules of the application should be provided: generally the source code documentation satisfy this need. During the coding each method and each data field has been commented in order to clarify its purposes: then tools that automatically retrieve the comments and put them into a single document have been used. In the Android version of the application it has been used Javadoc [25], while for the desktop one it has been used Doxygen [26]. In the appendix of the manual it has been put the explanation of the solutions adopted in some problems, like the ones in Section 6.1.1: these are particularly important since a maintainer has to be aware of the inner mechanisms of the application in order to understand the usefulness of some components and both the manual and the source code documentation don't consider this aspect.

Chapter 7

Utilization

At the end of the development of both versions of BlueFall, the software has been heavily used in order to obtain some data on which the efficiency and the effectiveness of the sharing schemes coupled with a wireless protocol could be evaluated. From one side it allowed to understand if the BlueFall ideas can be used to resolve the server-client model problems and from the other one it allowed to measure the goodness of the designed sharing schemes.

7.1 The data collection

The project of the data collection can be divided in two parts: the data collection on Android and the data collection on desktops/laptops. The data collection on Android has been performed on Samsung GT-i9250 devices (Figure 7.1a), which has been made available by the university: the version of the Android operating system which is installed in these device is 4.0 while the version of the Bluetooth protocol is 3.0. The other data collection has involved an heterogeneous set of computers (laptops and desktops) each one equipped with:

- a Trust Bluetooth 3.0 USB adapter (item number 17772, EAN-code 8713439177725) (Figure 7.1b);
- a Netgear Wireless-N 300 USB adapter (WN111-1VCNAS) (Figure 7.1c).

The Bluetooth adapter has been chosen since it supports the same version of Bluetooth supported by the mobiles used in the Android platform experiments. The WiFi adapter has been chosen since WPA supplicant, the software which allows to perform WiFi-Direct connections, can works only with a limited number of WiFi adapter models and the one chosen belongs to the set of working devices (the entire list of supported devices can be found in [27] and in [28]).

The experiments have been performed involving 3 devices and involving 5 devices: this offered the possibility to evaluate how much the "wireless protocol - sharing scheme" couple is able to support the adding of other devices and how much the performances degrade consequentially. In each experiment the device whose role was transmitter or first node downloaded a file from Internet and shared it with the other participants of the network. The file shared is a deb package whose size is about 10 megabytes: it has been chosen since it is not a heavy resource but it allows to see the behaviour of the network anyway (the sharing is not too much fast). In mobile

(a) Samsung GT-i9250

(b) Trust Bluetooth USB adapter

(c) Netgear WN111-1VCNAS

Figure 7.1: Main tools used in experiments

experiments, the file has been retrieved using a WiFi connection to a classic wireless access point, while in the other set of experiments the file has been retrieved using an Ethernet connection. This difference is due to the fact that mobiles and desktops can download heavy resources in different ways: while mobiles can only rely on WiFi to download big files (operating system updates, for example), desktops can use Ethernet connection, which is faster and less error-prone. Moreover in a network composed by desktops (a computer classroom, for example) it is quite common to find the Internet access provided by Ethernet cable. The experiments touched each one of the offered combinations between wireless protocols and sharing schemes. In particular, each one of the configurations measured in the data collection is uniquely identified by four variables:

- the platform, binary variable (values: Android, desktop/laptop);
- the number of devices involved, binary variable (values: 3, 5);
- the wireless protocol used, binary variable (values: Bluetooth, WiFi-Direct);
- the sharing scheme applied, binary variable (values: star sharing scheme, waterfall sharing scheme).

So the data collection has looked over 16 different configurations. For each configuration about 20 tests have been performed, then the tests results have been averaged in order to gain an idea of the way of working of the configuration. To easily gain 20 tests of each configuration, it has been exploited the possibility to share multiple resources: it has been sufficient to prepare a list of resources in which each entry matched the same resource. In this way just one set up of the configuration has been needed and devices have kept to gather data alone. Each test has provided in output a XLS file from each one of the network participants: BlueFall has been configured to store 10 samples on the sharing, so the results gained show the behaviour of the network after having shared about 10% of the resource, after having shared about 20% of the resource, and so on. The fact that measurements are performed on each participant of the network allows to evaluate the goodness of a configuration from different points of view. Each set of tests provides data on:

- total time involved;
- instantaneous data rate;
- average data rate;

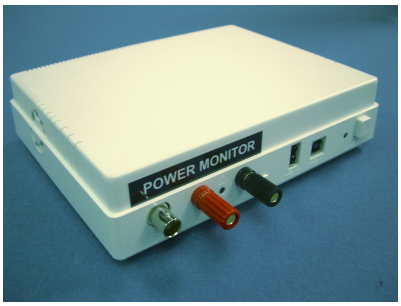
- standard deviation of data rate.

Each one of them provides an evaluation of the configuration from a different aspect. The time involved and the average data rate can be seen as the description of how fast is the configuration: generally an inverse relationship between the two measures can be identified since a sharing performed in few time implies a high data rate protocol necessarily. The instantaneous data rate is a snapshot of the data rate: if it is very similar to the average data rate, generally it implies a low standard deviation of the data rate, while if it differs a lot from the average data rate it generally implies a high standard deviation of the data rate. However a single instantaneous data rate value is not so significant: it becomes useful when it is observed during a sharing on the screen of the device since it traces each change of the data rate. The standard deviation of data rate is an extremely useful value since it consists in a measure of the instability of the connection: transmissions with a high value of standard deviation corresponds to hiccups transmissions while low values of standard deviation corresponds to stable transmissions. Assuming that a hiccups transmission is due to retransmissions, the standard deviation can be seen as a measure of how much busy is the wireless channel.

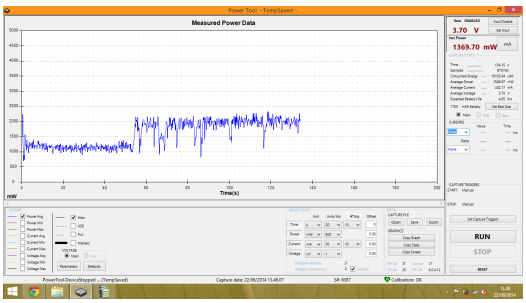
BlueFall provides also a way to measure the battery consumption of the device on which BlueFall is running: however this measurement is quite rough since it simply consists in the difference between the battery level recorded at the beginning of the sharing and the battery level recorded at the end of the sharing. This result may be heavily influenced by the other tasks that the device is performing and it generally represents a too much coarse-grained measurement: unless this measurement is performed on very long sharings, it cannot be used as a valid energy consumption measure. Since energy consumption is an important aspect on mobile platform, some external tools have been used to gain reliable measurements during the data collection. Initially a software tool has been tried, named PowerTutor [29]: it is an interesting application that allows also to understand the energy consumption due to a specific application which is running. However the application is designed for some specific models of mobiles and it presents some compatibility problems with other devices: for example, it was not able to identify the energy consumption due to the usage of WiFi device on the phones used in the experiments. For this reason, PowerTutor has been discarded. An alternative way to measure battery consumption was provided by an external hardware tool whose name is PowerMonitor [30]. The tool is designed to put itself in place of the energy provider of the device to measure: in this way it is able to measure the energy which is requested by the device in every moment. PowerMonitor has been provided by the university and it has been used on the Samsung phones of the experiments, while it hasn't been used on desktops or laptops since the energy consumption on desktop platform is not so much interesting.

The measurements of the energy consumption have been preceded by the identification of the energetic profiles, which corresponds to roles assigned to the devices depending on the energy consumptions performed. To identify these profiles, a classification of the possible causes of energy consumption is needed: each cause is labelled since it helps to define the energetic profiles in a second moment.

- C1: use of WiFi to download a file;
- C2: use of Bluetooth to receive a file;
- C3: use of Bluetooth to transmit a file to a single device;
- C4: use of Bluetooth to transmit a file to other 2 devices;



(a) The PowerMonitor tool



(b) A screen of PowerMonitor software

Figure 7.2: The PowerMonitor tool instrumentation

- C5: use of Bluetooth to transmit a file to other 4 devices;
- C6: use of WiFi-Direct to receive a file;
- C7: use of WiFi-Direct to transmit a file to a single device;
- C8: use of WiFi-Direct to transmit a file to other 2 devices;
- C9: use of WiFi-Direct to transmit a file to other 4 devices;

An energetic profile can be seen as a combination of these causes. Table 7.1 shows all the profiles that can be identified using BlueFall. It is important to note that there is no perfect matching between energetic profiles and roles that devices can assume depending on the sharing scheme: as the table shows, more than one role can match a single energetic profile and also more than one energetic profile can match a single role.

Profile #	Consumptions	Description
1	C1 + C4	It matches the device which acts as transmitter in a star sharing scheme composed by 3 devices with Bluetooth used as wireless protocol.
2	C1 + C8	It matches the device which acts as transmitter in a star sharing scheme composed by 3 devices with WiFi-Direct used as wireless protocol.
3	C1 + C5	It matches the device which acts as transmitter in a star sharing scheme composed by 5 devices with Bluetooth used as wireless protocol.
4	C1 + C9	It matches the device which acts as transmitter in a star sharing scheme composed by 5 devices with WiFi-Direct used as wireless protocol.
5	C1 + C3	It matches the device which acts as first node in a waterfall sharing scheme with Bluetooth used as wireless protocol. It matches also the device which acts as transmitter in a star sharing scheme composed by 2 devices with Bluetooth used as wireless protocol.

6	C1 + C7	It matches the device which acts as first node in a waterfall sharing scheme with WiFi-Direct used as wireless protocol. It matches also the device which acts as transmitter in a star sharing scheme composed by 2 devices with WiFi-Direct used as wireless protocol.
7	C2 + C3	It matches the device which acts as an intermediate node in a waterfall sharing scheme with Bluetooth used as wireless protocol.
8	C6 + C7	It matches the device which acts as an intermediate node in a waterfall sharing scheme with WiFi-Direct used as wireless protocol.
9	C2	It matches the device which acts as last node in a waterfall sharing scheme with Bluetooth used as wireless protocol. It matches also the device which acts as a receiver in a star sharing scheme with Bluetooth used as wireless protocol.
10	C6	It matches the device which acts as last node in a waterfall sharing scheme with WiFi-Direct used as wireless protocol. It matches also the device which acts as a receiver in a star sharing scheme with WiFi-Direct used as wireless protocol.

Table 7.1: Energetic profiles

To measure the energy consumption of the device the single sharing was considered as experimental unit: the evaluation of the energy consumption started once also the sharing started and it ended when also the sharing ended. In this way it was possible to see the variations of the energy consumption in relation to the task performed by BlueFall through the chart produced by the PowerMonitor software. However, in order to gain a single comparable value for each test, it has been chosen the average power consumption computed from the beginning to the end of the sharing.

Before starting to measure the energetic profiles identified, a baseline value is needed: this allows to understand if the energy consumption due to the usage of BlueFall is comparable with a normal usage of the phone or not. To obtain the baseline value, a baseline configuration has to be defined: it consists in the following elements.

- The brightness of the screen has been set to the maximum value.
- The background of the device has been set with a custom defined image. It consists in a solid color background whose specifications are:
  - Hue: 0.
  - Saturation: 5.
  - Value: 33.
  - Red: 85.
  - Green: 81.
  - Blue: 81.

  - HTML code: 555151.

- All the communication devices (WiFi, Bluetooth, 3G/HSDPA, NFC) have been switched off.
- No applications are active in background.
- The automatic obscuration of the screen has been delayed after 10 minutes of inactivity.
- The energy consumption value is considered the average power consumption after 10 minutes of inactivity.

A snapshot of the baseline evaluation can be seen in Figure 7.3: the chart clearly shows that the device is not performing any heavy task since the power consumption presents very small oscillations only. The voltage and the instantaneous power consumption can be seen on the right of the image written in red: since the power consumption is very stable, the average power consumption (which corresponds to the value assumed as baseline value) does not differ very much from the instantaneous power consumption.

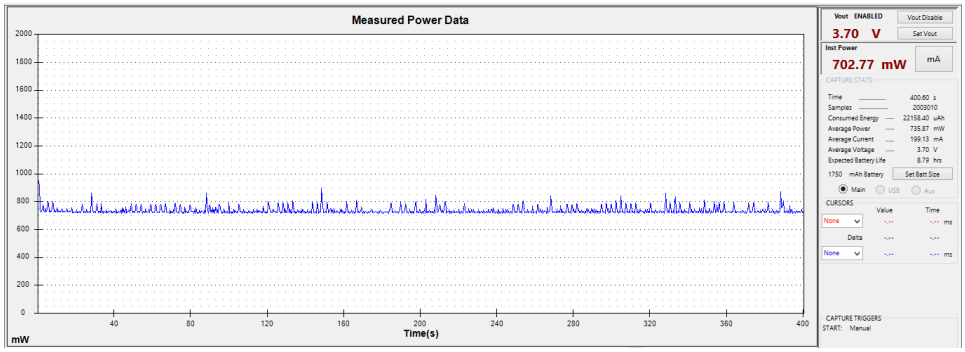


Figure 7.3: Chart showing the power consumption baseline evaluation

7.2 Problems during data collection

The data collection was not performed without difficulty, especially the data collection which concerns Android mobiles. However it was to be expected since the application of the sharing scheme does not match a traditional usage of the wireless protocols. Unfortunately the occurrences of these problems are not rare and they are generally not recoverable from the sharing scheme point of view (the application does not fail, but the connections do not work any more).

Using the Bluetooth protocol on Android mobiles, the most common error is the "broken pipe" one: it matches an IOException and it generally happens when a device tries to communicate through a socket with another device which has already closed the connection for some reasons. The device which has closed the connection too early is not passed through the classic socket closure, but the program or the device has failed and this causes the break up of the connection. Since BlueFall has never failed (unexpectedly stopped) during the data collection, the causes of the problem should be searched in the Bluetooth device working.

Listing 7.1: Broken pipe error example on Android

```
E/bt-btif(1786): recv error, errno:11, fd:65, size:0, received:-1
E/(1786): ## ERROR : PORT_WriteDataCO: p_data_co_callback
    DATA_CO_CALLBACK_TYPE_OUTGOING failed, available:0##
W/bt-btif(1786): invalid rfc slot id: 3
W/System.err(1749): java.io.IOException: Broken pipe
W/System.err(1749):     at android.net.LocalSocketImpl.writeba_native(
    Native Method)
W/System.err(1749):     at android.net.LocalSocketImpl.access$600(
    LocalSocketImpl.java:29)
W/System.err(1749):     at android.net.
    LocalSocketImpl$SocketOutputStream.write(LocalSocketImpl.java:126)
W/System.err(1749):     at android.bluetooth.BluetoothSocket.write(
    BluetoothSocket.java:424)
W/System.err(1749):     at android.bluetooth.BluetoothOutputStream.write
    (BluetoothOutputStream.java:85)
W/System.err(1749):     at com.mpozza.bluefall.model.connect.
    BluetoothConnector.writeData(BluetoothConnector.java:196)
W/System.err(1749):     at com.mpozza.bluefall.controller.threads.
    Transmitter.run(Transmitter.java:58)
```

The main problem is that the potential causes of this type of error are a high number and the log files of the application are not able to report errors which do not concern the code: they can only show that a problematic situation happen (the exception) but they do not know the inner causes of the problem. The exception, in its most general form, is quite common in networking programming since it concerns each type of socket, but the specific "broken pipe" error due to the usage of Bluetooth on Android platform is almost unknown. It is possible to note the error using BlueFall looking at the progress bar which concerns the output connection: if it is blocked from a considerable amount of time, then probably the error happened. The most annoying aspect of this error is that it also causes Bluetooth to be not manageable any more on the device in which the error appears: switching Bluetooth on or off through the classic "Settings" menu of Android will not work any more. The only solution is a restart of the operating system, which will restore the classic Bluetooth functionality. The occurrences of the error which have been observed are:

- Star sharing scheme on 3 devices: 2 occurrences over 39 sharings (5,13%);
- Star sharing scheme on 5 devices: 2 occurrences over 25 sharings (8%);
- Waterfall sharing scheme on 3 devices: no occurrences over 40 sharings (0%);
- Waterfall sharing scheme on 5 devices: 2 occurrences over 17 sharings (11,76%).

So the error is quite rare from the single sharing point of view: the bad aspect is the need to restart the device once it happens.

Another problem which regards the Bluetooth protocol on Android platform is the "command timeout" one. It is very rare, but it brings as consequence the same ones of the "broken pipe" error: the sharing scheme fails and the device has to be restarted in order to gain the control of Bluetooth device again. Also this error is almost unknown so no causes and no solutions have been found. Both of the problems described seem related to the intensity of use of the Bluetooth device in some way: in fact, Bluetooth device is an I/O device, so it is natured "slow", and it may be used from two threads at the same time. An hypothesis is that Bluetooth device performs its work in a "delayed" way: when a piece of code requests an operation with Bluetooth device, the Bluetooth device immediately accepts the work, so the call

can return to the requester, and then it starts to perform the requested operations. If too much calls are requested and device is busy, a request which has no answer could be translated to a "command timeout" error or to the fact that the connection is compromised somehow (the "broken pipe" error). However this is only an hypothesis and it is not confirmed at present by any fact.

Listing 7.2: Command timeout error example on Android

```
WARNING : BTU HCI(id=0) command timeout. opcode=0x406
HCI Cmd timeout counter 1
```

The main problem which regards WiFi-Direct on Android platform is the "socket timeout" problem: it concerns the connection performed to download the resource and it consists in an attempt to keep downloading the resource once the connection has fall down, causing the error. It is easy to note in BlueFall use since the WiFi icon, which is normally present to symbolize that a WiFi Internet connection is active, suddenly disappears (Figure 7.4).

Figure 7.4: The WiFi Internet connection icon, surrounded with a red square

So the "socket timeout" problem is shown as a consequence of the fact that the Internet connection has been broken for unknown causes: an hypothesis is that the simultaneous usage in downloading the resource and in sharing the resource using WiFi-Direct represents a too much heavy burden for the WiFi device, just like in Bluetooth case. The occurrences of the error which have been observed are:

- Star sharing scheme on 3 devices: 3 occurrences over 27 sharings (11,11%);
- Star sharing scheme on 5 devices: 2 occurrences over 13 sharings (15,38%);
- Waterfall sharing scheme on 3 devices: 2 occurrences over 32 sharings (6,25%);
- Waterfall sharing scheme on 5 devices: 2 occurrences over 11 sharings (18,18%).

As in the "broken pipe" error of the Bluetooth device, the less error-prone configuration is the waterfall scheme on 3 devices while the worst one is the waterfall scheme on 5 devices. The two star configurations show a worsening once the number of participants to the network increases.

In the desktop version of the application no one of the errors presented appeared and, in general, the sharings have been never stopped for any reason. The only stops have been due to automatic sleeping of the computers after a certain amount of time, which caused the sharing to be paused: it has been sufficient to configure the energy saving properties of the operating system to avoid this nuisance. The fact that the desktop version of the application showed no problems may be due to the fact that the application is written in the same language of the software which handles the Bluetooth and WiFi devices while in Android version the Java code performs calls to "native" methods, which are methods written in other languages (C in this case): this indirection in the devices management could be identified as the cause of the problems explained, but this is an hypothesis which is not supported by any fact.

A different type of problem was found using the desktop version of BlueFall: the devices had difficulties to find each other using WiFi-Direct. While using mobiles there

were no problems, using WiFi-Direct USB devices on desktops/laptops the computers had to stay in a range of about 1 meter in order to allow the search process to give some results. In a context like this the goodnesses of WiFi-Direct cannot be compared with the discomfort of the constrain described: however, since with mobiles the ranges are normal, the problem must stay in the WiFi devices used with desktops. An hypothesis is that the problem is a design one: while the antenna of the mobiles used had been designed also to support WiFi-Direct protocol, in the desktop case the WiFi devices used are WiFi USB adapters on which WiFi-Direct works, but they have not been designed to support this protocol. These devices have been designed and commercialized before the spread of WiFi-Direct protocol, but since they work with a driver which supports WiFi-Direct, the protocol manages to work on them. So when these devices are asked to create a WiFi-Direct group they have difficulties to make other devices find them since they have not been designed "to be found" but "to find" instead: this is manifested through the difficulties to find other WiFi-Direct devices with normal ranges.

Chapter 8

Results and considerations

The results of the data collection are firstly separated by platform (Android and desktop), then they are grouped depending on the number of devices involved in the network. This allows to evaluate the different combinations of wireless protocol and sharing scheme keeping the same number of devices. The results presented are took from a device which acts as a receiver (if star sharing scheme is used) or as last node (if waterfall sharing scheme is used): in this way the evaluation of the total time involved is the best possible (in waterfall configurations nodes in the first part of the waterfall may finish to do they job before each one in the waterfall has managed to obtain the resource). Moreover the data gathered on data rate concern the incoming connection (the connection which brings data to the device): this is preferred than performing measurements on the outcoming connection since this one may be lightly influenced by the fact that there are more than one device to send data.

8.1 Results on Android platform

The section contains the results gathered through the usage of BlueFall on Android phones. The results are separated depending on the number of devices involved, while the data which regard energy consumptions are gathered in the last section.

8.1.1 Results with 3 devices

The section contains the data gathered applying the available combinations of wireless protocols and sharing schemes on 3 devices. The results are observed from each one of the different aspects offered: time involved, average data rate and standard deviation of data rate. The instantaneous data rate is not considered, except for limited contexts, since it represents the less significant part of the data collection.

8.1.1.1 Time comparison

The chart in Figure 8.7 shows some predictable results: the configurations which use WiFi-Direct request less time than the ones which use Bluetooth. In particular, Bluetooth configurations ask about twice the time requested by the WiFi-Direct ones. It is interesting to note that the waterfall sharing scheme gives the best results using WiFi-Direct while it requires more time than the star configuration to perform the sharing using Bluetooth. However, looking to the sharing schemes which use the same

wireless protocol, the differences between the results are not big, since the highest difference recorded is 10 seconds: anyway, the biggest difference is recorded at the end of the sharing and since the trends of the configurations are quite regular (they look very much like straights) it seems that the more heavy is the resource to share, the bigger is the final difference between the sharing schemes.

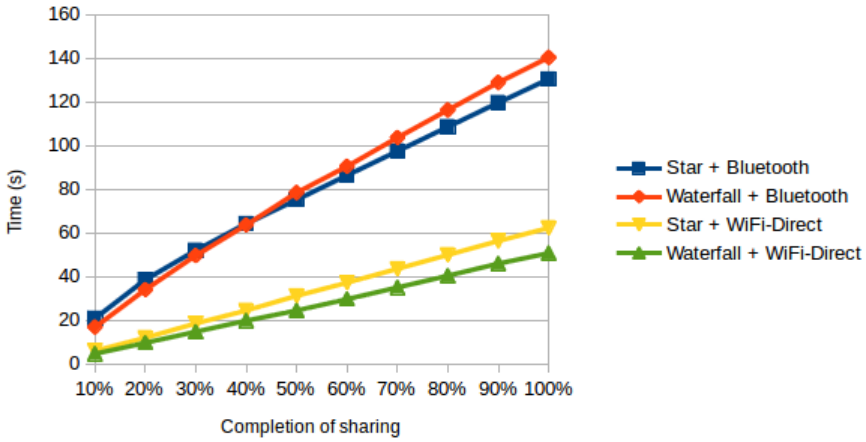


Figure 8.1: Time comparison of the configurations with 3 devices (mobile)

8.1.1.2 Average data rate comparison

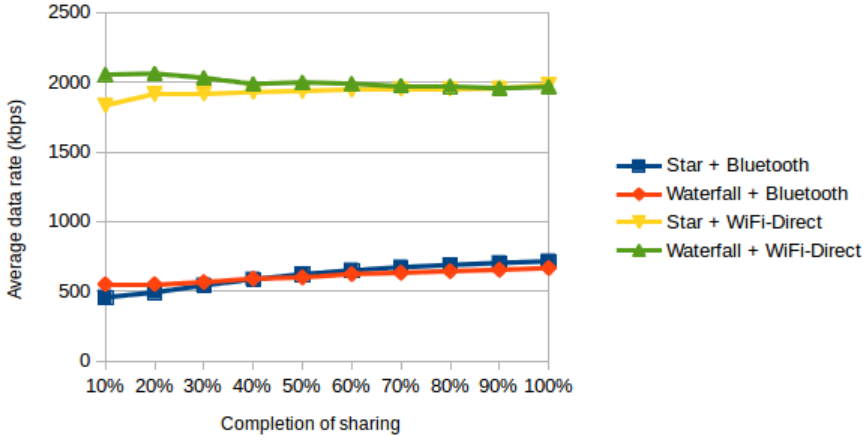


Figure 8.2: Average data rate comparison of the configurations with 3 devices (mobile)

Figure 8.2 shows that there is a ratio of about 4:1 between the average speed of WiFi-Direct and the one of Bluetooth. Considering the same wireless protocol, the sharing schemes behave in an almost equal manner since the differences are really small: except

for the begin, the behaviour of the waterfall sharing scheme is identical to the one of the star sharing scheme using WiFi-Direct, while there are just insignificant differences between the two configurations using Bluetooth.

8.1.1.3 Standard deviation of data rate comparison

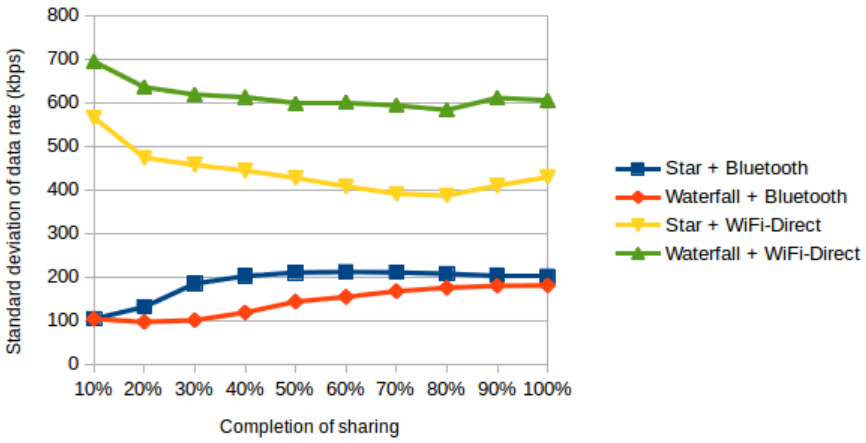


Figure 8.3: Standard deviation of data rate comparison of the configurations with 3 devices (mobile)

The chart in Figure 8.3 is certainly the most interesting of the results using 3 devices: the waterfall sharing scheme coupled with Bluetooth protocol seems the most stable configuration since the the standard deviation settles between 100 and 200 kbps, even if it keeps growing till the end of the sharing. The star sharing scheme coupled with Bluetooth protocol has an almost constant standard deviation of 200 kbps for the biggest part of the sharing. The two configurations which use WiFi-Direct differ a lot from the standard deviation point of view even if both of them trace a similar behaviour: they start with a high standard deviation and they gradually reduce it till a quite stable value is found. The two configurations always keep a difference of about 150 kbps, with peaks of 200 kbps: while the standard deviation of the waterfall configuration shifts between 700 and 600 kbps, the one of the star configuration goes from a bit less than 600 kbps to a bit less than 400 kbps.

To decree the most stable configuration it is not sufficient considering the configuration with the lowest standard deviation: in fact this value must be compared with the average data rate to gain a measure of how much great are the oscillations from the medium value. For both the Bluetooth configurations the standard deviation corresponds to about $\frac{1}{3}$ of the average data rate. Using WiFi-Direct and considering the average data rate value of the two configurations (about 2000 kbps), in the waterfall configuration the standard deviation is about $\frac{1}{3}$ of the average data rate while in the star configuration the same value goes from about $\frac{1}{3}$ to $\frac{1}{5}$. So the most stable couple seems to be the star sharing scheme with WiFi-Direct.

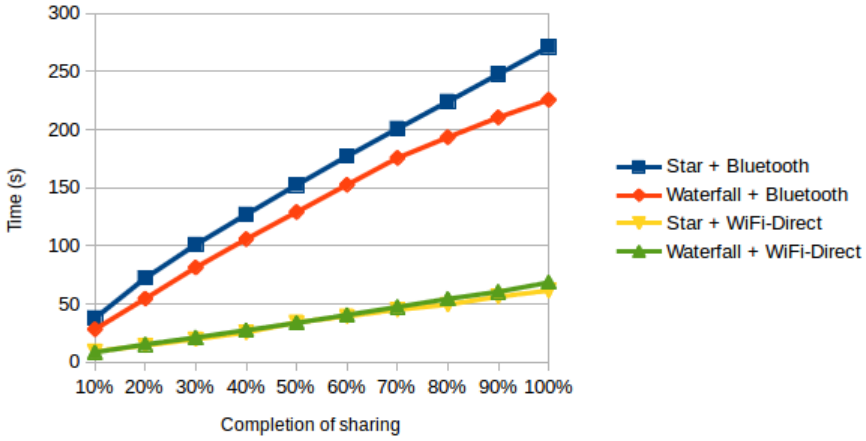


Figure 8.4: Time comparison of the configurations with 5 devices (mobile)

8.1.2 Results with 5 devices

The section contains the data gathered applying the available combinations of wireless protocols and sharing schemes on 5 devices.

8.1.2.1 Time comparison

The chart in Figure 8.4 shows a turnaround considering the previsions made in the configurations with 3 devices: the waterfall sharing scheme coupled with Bluetooth requests less time than the star sharing scheme with the same wireless protocol while in the configurations with 3 devices happens just the opposite. This can be interpreted as a signal of the difficulties that the transmitter device is encountering in the star sharing scheme once the receivers doubled. The two configurations which use WiFi-Direct show an almost identical behaviour: comparing these results with the ones in chart 8.1 appears that the waterfall configuration has sightly worse performances while the star configuration shows almost the same behaviour of the case with 3 devices. It seems that the adding of two devices is irrelevant from the star sharing scheme point of view: since in Bluetooth case happens just the opposite, the merits goes to WiFi-Direct protocol. Comparing the total time involved, the two Bluetooth configurations differs of about 40 seconds, which is a considerable amount of time: from the time point of view, the waterfall scheme coupled with Bluetooth is better than the star sharing scheme with the same wireless protocol. It is interesting to note that the doubling of the devices has brought also the doubling of the difference between the total time requested by the Bluetooth configurations and the one requested by the WiFi-Direct configurations: while the WiFi-Direct configurations remained almost indifferent to the adding of the devices, the Bluetooth ones had significantly worsen their performances.

8.1.2.2 Average data rate comparison

Figure 8.5 first shows that the waterfall sharing scheme coupled with Bluetooth works better than the star sharing scheme with the same wireless protocol: this not only re-

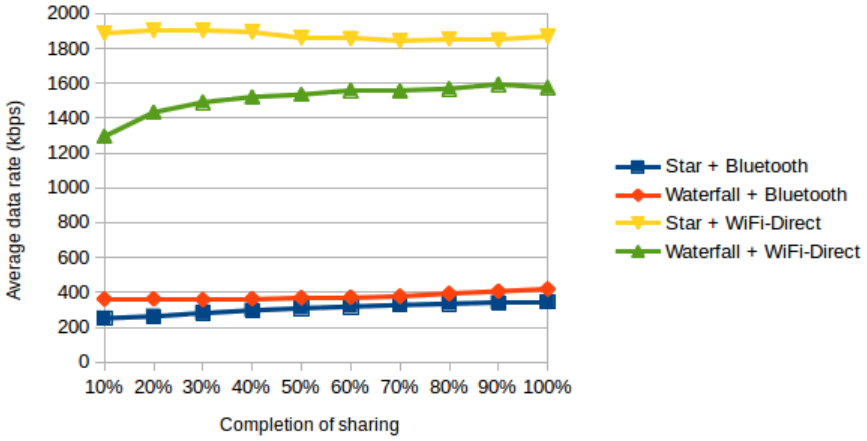


Figure 8.5: Average data rate comparison of the configurations with 5 devices (mobile)

flects the time results shown in Figure 8.4 but it also reflects the original aims for which the waterfall sharing scheme was designed since it was though as a solution for a star sharing scheme "too much crowded". Looking at the results of the WiFi-Direct configurations, the star sharing scheme behaves better than the waterfall sharing scheme: the average data rate settles between 2000 kbps and 1800 kbps, while the average data rate of the waterfall configuration goes from about 1600 kbps to about 1200 kbps. It seems that a centralized approach fits better than a distributed one using WiFi-Direct, while in Bluetooth configurations is valid the opposite. Comparing the results with the ones using 3 devices, every configuration has worsen its performances: the Bluetooth configurations have lost at least 200 kbps, while WiFi-Direct configurations have lost from 100 kbps to 600 kbps. So the two Bluetooth configurations have deteriorated by about $\frac{1}{3}$ their performances, the waterfall sharing scheme with WiFi-Direct has deteriorated by about $\frac{1}{4}$ ($\frac{1}{3}$ in the some moments) its performance and the star sharing scheme with WiFi-Direct has deteriorated by less than $\frac{1}{10}$ its performances. From this point of view, the configuration which works better is the star sharing scheme with WiFi-Direct.

8.1.2.3 Standard deviation of data rate comparison

The image in Figure 8.6 shows that, from the Bluetooth usage point of view, the waterfall configuration always keeps a higher standard deviation compared to the star sharing scheme one. So it seems that the fact that the waterfall configuration works better (less time, higher data rate) is compensated by the fact that it has a higher standard deviation of data rate, signal that the connection is quite unstable. However, looking at the ratio between the standard deviation and the average data rate, the two configurations work with the same stability more or less. In fact in waterfall configuration the standard deviation goes from 100 kbps to less than 200 kbps, while the average data rate is about 400 kbps: considering a medium value of 150 kbps as standard deviation, the ratio is between $\frac{1}{3}$ and $\frac{1}{2}$. The star sharing scheme has an almost constant standard deviation of 100 kbps, while the average data rate goes from 200 kbps to 300 kbps, so its ratio varies in the same range of the waterfall

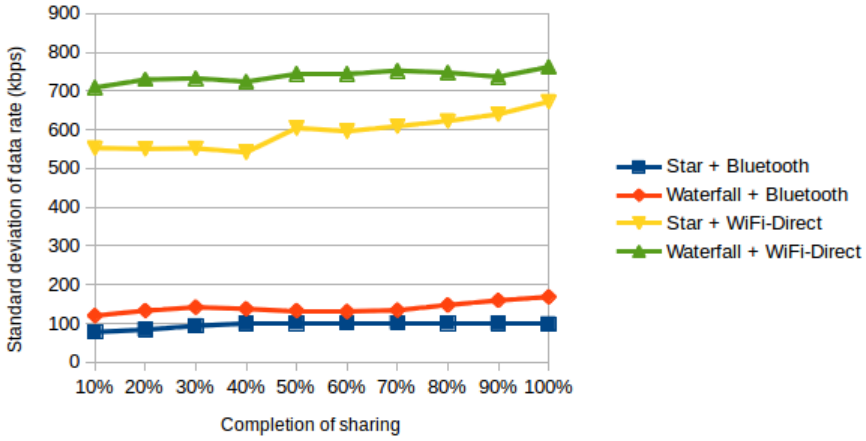


Figure 8.6: Standard deviation of data rate comparison of the configurations with 5 devices (mobile)

configuration. Comparing the results with the ones using 3 devices, it is important to note that the star sharing scheme has decreased its standard deviation (from 200 kbps to 100 kbps), while the waterfall sharing scheme has kept almost the same behaviour.

With the usage of WiFi-Direct, the star sharing scheme seems to work better than the waterfall one: it has a standard deviation which goes from 500 kbps to 700 kbps, while the waterfall sharing scheme has a standard deviation of about 700 - 750 kbps. Looking at the averages in Figure 8.5 the ratio between the standard deviation and the average data rate is certainly lower in the star sharing scheme since it always has an average data rate higher than the waterfall sharing scheme. It is interesting to note that the standard deviations of the WiFi-Direct combinations trace almost the same behaviour both with 3 devices and with 5 devices since the standard deviation of the waterfall sharing scheme is always higher than the one of the star sharing scheme. However the standard deviations generally increases with the adding of new devices: since the average data rates decrease concomitantly the same event, the adding of new devices produces generally a degradation of the performances.

8.1.3 Energy consumptions comparison

The chart presents the power consumption recorded in baseline evaluation (green) an then it shows the couples of energetic profiles which changes only for the wireless protocol used: the blue columns correspond to the Bluetooth energetic profiles, while the red ones correspond to WiFi-Direct energetic profiles. The explanation of the energetic profiles can be found in Table 7.1.

Figure 8.7 shows some unexpected results. In fact it was expected that the low data rate of Bluetooth protocol was compensated by a low energy consumption since the last one corresponds to one of the main purposes in protocol designing: instead it seems that Bluetooth energetic profiles always consume more than the WiFi-Direct ones, except for the last couple of profiles. An hypothesis on this strange results is that Bluetooth protocol is being used in a wrong way compared to the goals for which it has been designed: in fact Bluetooth was designed as a cable replacement protocol,

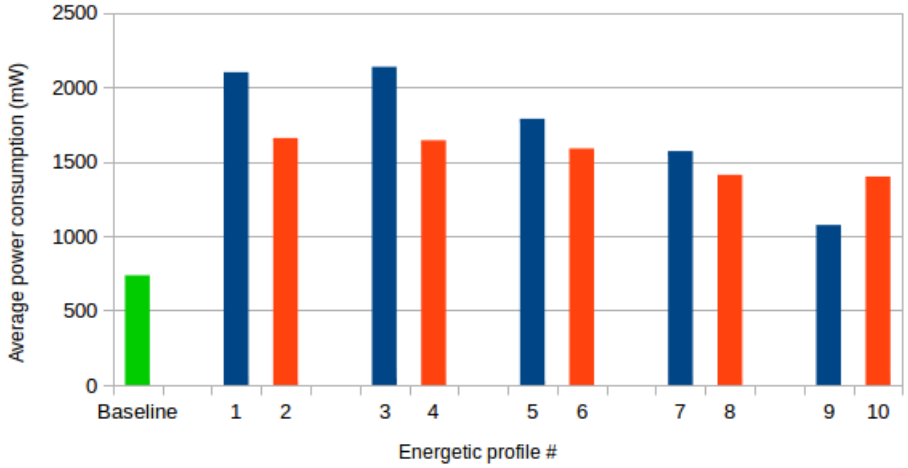


Figure 8.7: Comparison on energetic consumption between the energetic profiles

for short and few data transmissions, while in BlueFall it is used intensively for long periods and this incorrect usage of the protocol may lead to the annihilation of all of its power saving properties. WiFi-Direct instead has been designed to bring the WiFi speed (802.11b/g/n) in P2P communications, so it is suited perfectly to the needs of a protocol to be used in heavy data transmissions: these considerations can justify the results in chart 8.7.

The chart in Figure 8.8 confirms the results shown in Figure 8.7: in fact, the expected battery life values are in inverse ratio with the average power consumption ones, so the WiFi-Direct energetic profiles generally show a higher battery life than the Bluetooth ones. The only exception is for the couple of energetic profiles which considers the receiver role (or the last node role): here it seems that Bluetooth profile works better than the WiFi-Direct one from the power consumption point of view. This may be due to the fact that a device which assumes this role has just to listen to incoming data, so no transmissions have to be performed on the same hardware device and no downloads from Internet are needed: these elements make the Bluetooth protocol more suitable for the needs of the device since there are a lot of Bluetooth devices designed to only listen to incoming data (the device which listens to the inputs from mouse and keyboard, for example).

The charts which trace the average power consumption during all the sharing have allowed to understand the behaviour of the power consumption depending on some “external” events: Figure 8.9 shows 3 Bluetooth energetic profiles and they are particularly interesting since they clearly show the influence of the download of the resource to share in power consumption. The first chart shows a device whose energetic profile matches the profile #5, so it is the first node of a waterfall: the power consumption keeps to stay over 2000 mW till the resource is being retrieved, than it suddenly fall below 2000 mW and it stays there till the end of the sharing. This is a quite normal behaviour: the downloading of a file matches a higher power consumption then the transmission of the file alone.

The more interesting aspects are shown in the other two charts: it seems that

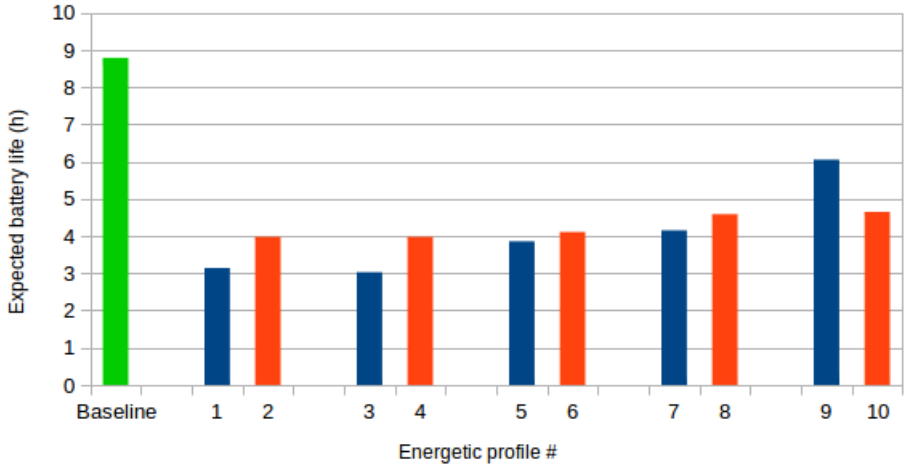
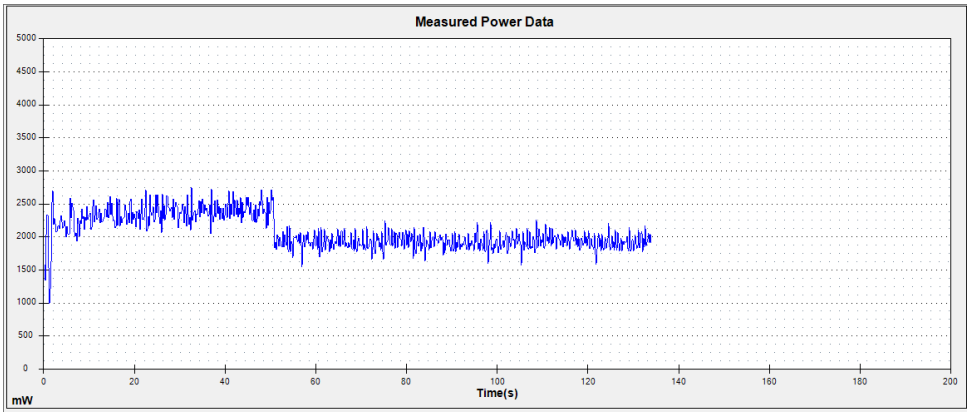


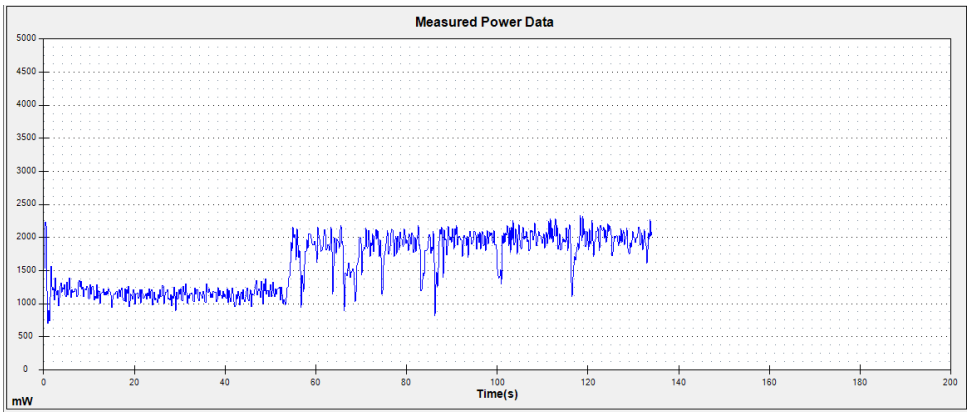
Figure 8.8: Comparison on expected battery life between the energetic profiles

the end of the download of the resource being shared brings the opposite effect if compared to the one brought on the first chart. In fact, at the end of the download, the average power consumption increases in the other two charts, in particular the increase is greater in the nearest devices, since profile #7 and #8 match respectively an intermediate node and the last node in waterfall. This behaviour may be due to the fact that, being the file already downloaded, the first device in the waterfall can now concentrate on the sharing of the file, so a greater interaction with the other devices is requested: this greater "involvement" of the other devices could be translated to a greater energy consumption, so after the file has been downloaded the sharing scheme "accelerate". Supporting the hypothesis there is the fact that WiFi device, which is used to retrieve the file, certainly interferes with Bluetooth device, so when the WiFi device is not needed any more the Bluetooth device can work better, that is manifested by the greater interaction with the other devices.

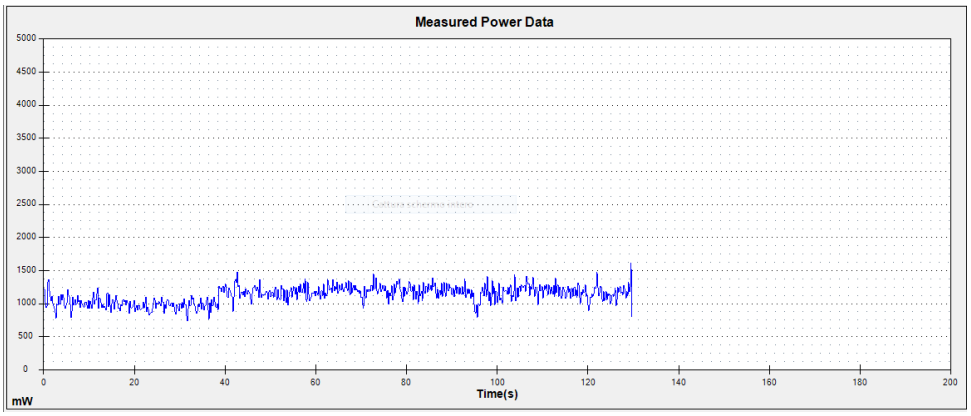
The charts relating the WiFi-Direct (Figure 8.10) show a more irregular behaviour: it is not possible to understand if some external events have happened simply looking them since the average power traces big oscillations during all the sharing. These oscillations are present regardless of the energetic profile: this justifies the fact that the results of the energetic profiles which use WiFi-Direct tend to be very similar among them, like in Figure 8.7 and 8.8. The instability of the average power consumption is strongly linked to the instability of the data rate: generally, it can be observed that the more the data rate is hight, the more the average power consumption grows. It is important to note that WiFi-Direct allows an almost instantaneous propagation of the resource being downloaded to the other devices since its speed is enough to compete with the speed of Internet connection: this means that it is not possible to distinguish the end of the download of the file since it almost coincides with the end of sharing.



(a) Profile #5

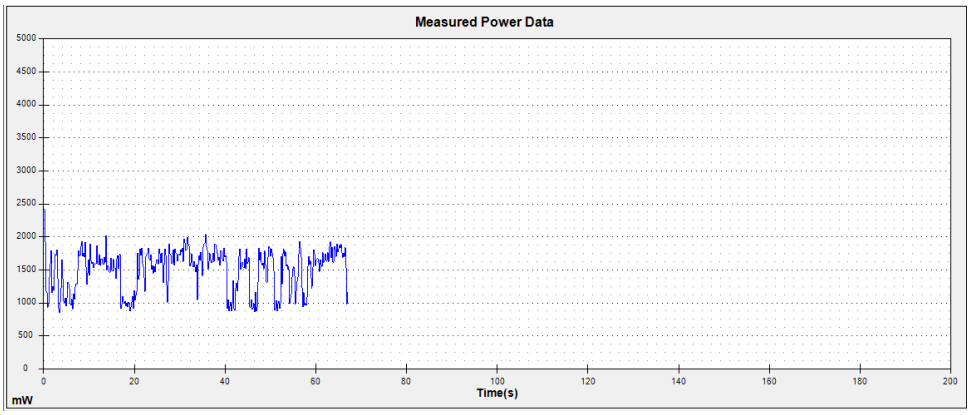


(b) Profile #7

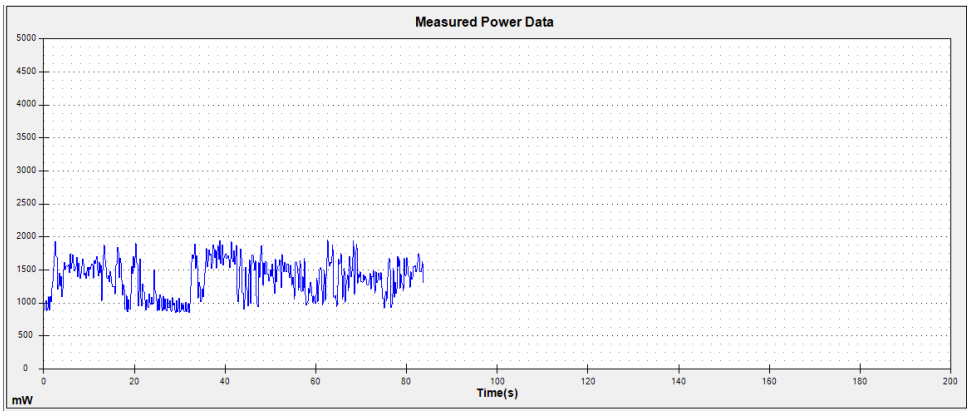


(c) Profile #9

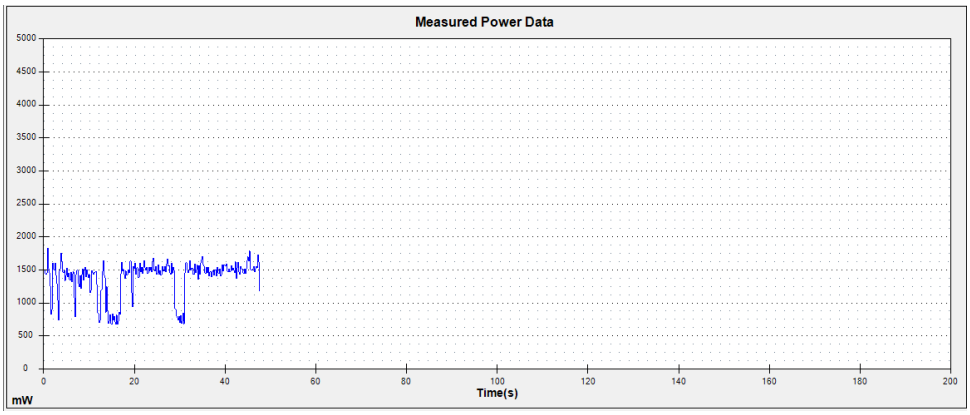
Figure 8.9: Behaviour of average power consumption on 3 profiles which uses Bluetooth



(a) Profile #6



(b) Profile #8



(c) Profile #10

Figure 8.10: Behaviour of average power consumption on 3 profiles which uses WiFi-Direct

8.1.4 Considerations on mobile experiments

From the data collection performed on Android platform some considerations can be performed: the first one is that WiFi-Direct is certainly a better solution compared to Bluetooth for the application of a sharing scheme. It is not only faster than Bluetooth, but it seems that WiFi-Direct generally consumes less than Bluetooth: moreover, since its speed allows an immediate spread of the resource in the network individuated by the sharing scheme, it can be used as a good solution for the server-client problems described. Considering the sharing schemes used with WiFi-Direct, the adding of new devices is born better by the star sharing scheme, which records a lower data rate loss and a lower standard deviation increase. An important aspect is the nature of the high standard deviation of the WiFi-Direct configurations: during the experiments it has been observed a lot of times that the data rate recorded a very hight value and then a very low or null value in the immediately following measurement, keeping to swing in this way during all the sharing. This can be due to the fact that WiFi-Direct is a fast protocol and allows an almost immediate propagation of the data retrieved by the "downloader" thread: this means that the thread which uses WiFi-Direct to transmit data towards other destinations is not able to accumulate data (unlike the Bluetooth situation) so it keeps to transmit big amount of data and then it waits for other data brought by the other thread. In few words, all the data that is retrieved by one thread is immediately propagated by the other thread, which has to wait for new data: this alternation between transmission and waits is the cause of the high standard deviation. Finally, from the time point of view there are no great changes between star sharing scheme and waterfall one: this means that also the other scheme can be used without problems considering that time consists in the only aspect that really interests to the final user.

Bluetooth solutions cannot compete with WiFi-Direct ones: however they can be useful in networks of old devices, in which WiFi-Direct is unsupported. The sharing schemes record almost the same performances on all the aspects considered, so it is difficult to decree the best Bluetooth sharing scheme. The only important result is that the adding of new devices is born better by the waterfall sharing scheme, which requests less time than the star sharing scheme to complete the sharing.

8.2 Results on desktop platform

The section contains the results gathered through the usage of BlueFall on desktops and laptops. Also in this case the results are fist separated by the number of devices, but energy measurements have not been performed since this aspect is not so important on this type of platform.

8.2.1 Results with 3 devices

The section contains the data gathered applying the available combinations of wireless protocols and sharing schemes on 3 devices.

8.2.1.1 Time comparison

The chart in Figure 8.11 shows that the configuration which asks more time to complete the work is the waterfall sharing scheme with the usage of Bluetooth. It requests about 50 seconds more than the star sharing scheme using the same wireless protocol,

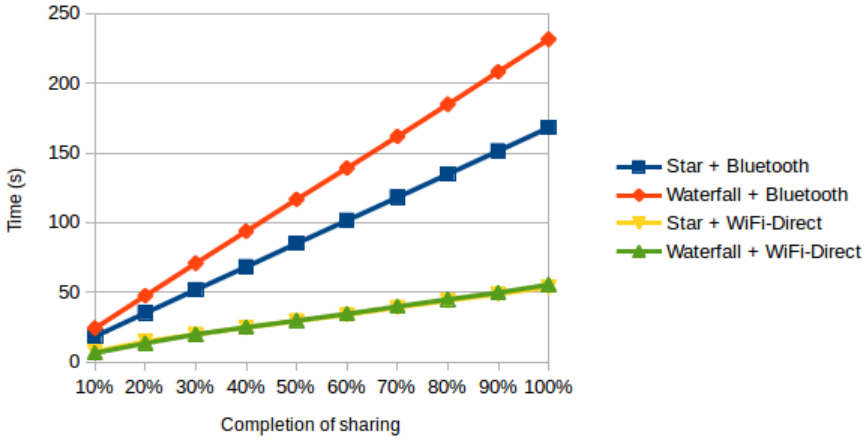


Figure 8.11: Time comparison of the configurations with 3 devices (desktop)

which is a considerable amount of time. The two WiFi-Direct configurations obtain the same results instead: they both request a bit more than 50 seconds to complete the job. It is interesting to compare these results with the ones in mobile environment (Figure 8.1): while the two WiFi-Direct configurations behave more or less in the same way, the two configurations which use Bluetooth differ a lot from the mobile results. In particular, the waterfall sharing scheme applied in desktop environment asks about 100 second more than the same configuration in mobile environment: since the sharing scheme is the same, the motivations of this strange result must be searched in the Bluetooth devices used.

8.2.1.2 Average data rate comparison

From the Figure 8.12 it appears that the two WiFi-Direct configurations behave in the same way, while using Bluetooth the star sharing scheme records a higher average data rate compared to the one of the waterfall sharing scheme. The difference between the two averages is of about 150 kbps: looking to the average values, this means that the star sharing scheme is more effective of about 30% from the waterfall sharing scheme point of view.

Comparing the results obtained with the corresponding ones on mobile platform it appears that, while the star sharing scheme coupled with Bluetooth behaves more or less in the same way in both of the platforms, all the other configurations show considerable differences. The waterfall sharing scheme coupled with Bluetooth always keeps an average data rate of 400 kbps on desktops, so it always record a difference with the same configuration on mobiles from 100 kbps to 200 kbps: it corresponds to an enormous difference since the mobile configuration is up to 50% faster than the desktop configuration. The configurations which use WiFi-Direct on mobile platform keep an average data rate of about 2000 kbps all the sharing long, while the two corresponding desktop configurations start from 1200 - 1400 kbps and gradually reach the maximum of 1800 kbps. In this case the difference between the two platforms is initially great but it is reduced as the end of the sharing is reached: finally, the desktop configurations are only 10% slower than the Android ones.

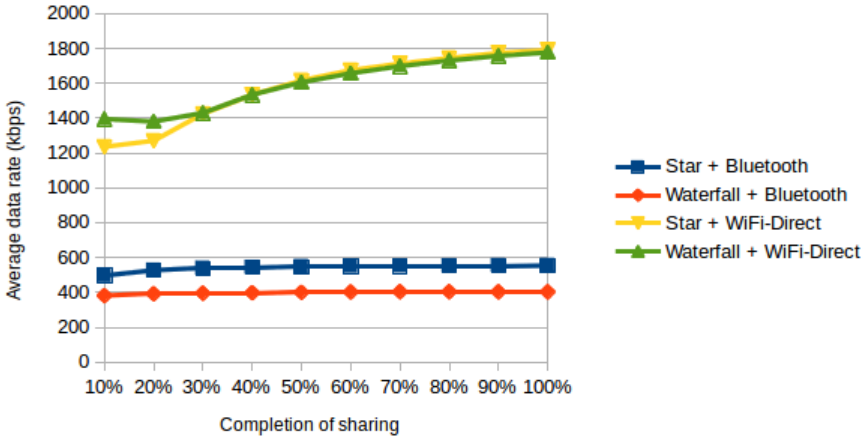


Figure 8.12: Average data rate comparison of the configurations with 3 devices (desktop)

8.2.1.3 Standard deviation of data rate comparison

The chart in Figure 8.13 shows that the configurations which use the same wireless protocol tend to have similar behaves: the two WiFi-Direct configurations show a strange "wave" behaviour, in which the standard deviation initially decreases, then increases, and then decreases again, while the two Bluetooth configurations keep decreasing the standard deviation slowly till a stable value is found. The ratio between the standard deviation and the average data rate is considered for each configuration.

- The star sharing scheme coupled with Bluetooth has an average data rate of about 550 kbps while it has a standard deviation which goes from more than 150 kbps to less than 100 kbps, so its ratio goes from less than $\frac{1}{5}$ to more than $\frac{1}{4}$.
- The waterfall sharing scheme coupled with Bluetooth has a ratio of about $\frac{1}{8}$ (except for the begin of the sharing, in which it is higher).
- The star sharing scheme with the usage of WiFi-Direct has an average data rate which goes from 1200 kbps to 1800 kbps and its standard deviation goes from 450 kbps to 550 kbps: averaging the values, the ratio is about $\frac{1}{3}$.
- The waterfall sharing scheme with the usage of WiFi-Direct has an average data rate which goes from 1400 kbps to 1800 kbps and its standard deviation goes from 400 kbps to 500 kbps: averaging the values, the ratio is less than $\frac{1}{4}$.

From this analysis, it seems that the waterfall sharing scheme couple with Bluetooth is the most stable configuration in this scenario, but since it also records the worst total time and the worst average data rate it cannot be considered a good configuration. The same scheme coupled with WiFi-Direct protocol seems the best solution at present since it records the same performances of the other WiFi-Direct configuration but it also records a lower standard deviation.

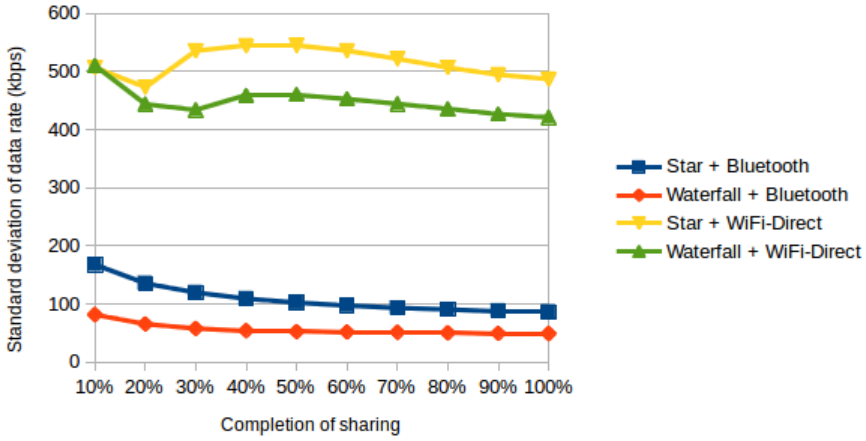


Figure 8.13: Standard deviation of data rate comparison of the configurations with 3 devices (desktop)

8.2.2 Results with 5 devices

The section contains the data gathered applying the available combinations of wireless protocols and sharing schemes on 5 devices.

8.2.2.1 Time comparison

From the chart in Figure 8.14 it appears that the WiFi-Direct configurations keep to have the same behaviour: just in the end the waterfall sharing scheme asks few seconds more than the star sharing scheme. The Bluetooth configurations reach more or less the same results even if in the scenario with 3 devices the waterfall sharing scheme behaves worse than the other one. This can be due to two factors:

1. the star sharing scheme has worsen its performances following the adding of new devices;

2. the waterfall sharing scheme has improved its performances following the adding of new devices.

The second hypothesis is really unlikely, while the first one is the favourite since the same happens in mobile environment: with 3 devices the configurations behaves in the same way, while with 5 devices the waterfall configuration behaves better. So this is another case in which the waterfall sharing scheme seems to resolve the problem of a star sharing scheme "too much crowded". It is important to note that the total time requested by the Bluetooth configurations has almost doubled, just like in the mobile environment, but while in mobile environment the ratio between the time requested by the WiFi-Direct configurations and the time requested by the Bluetooth configurations passes from $\frac{1}{2}$ to $\frac{1}{4}$, in desktop environment the same ratio passes from $\frac{1}{4}$ to $\frac{1}{8}$.

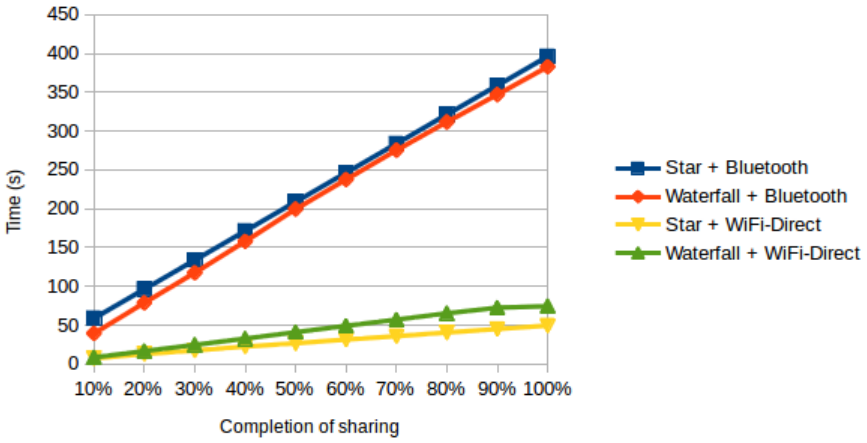


Figure 8.14: Time comparison of the configurations with 5 devices (desktop)

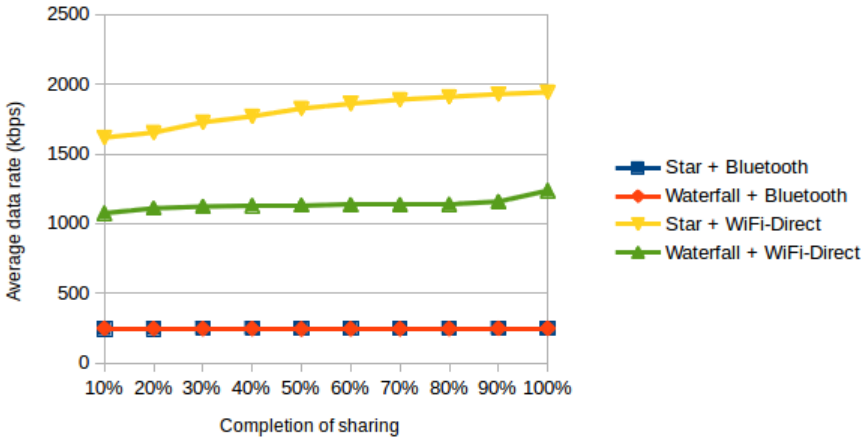


Figure 8.15: Average data rate comparison of the configurations with 5 devices (desktop)

8.2.2.2 Average data rate comparison

From the chart in Figure 8.15 it appears that the two Bluetooth configurations work exactly in the same way: they keep a data rate of about 250 kbps all the sharing long. Both of the configurations have worsen their performances with the adding of new devices, but the star sharing scheme has deteriorated more than the other one: this fact is concordant with the conclusions on the time in the previous section.

The two WiFi-Direct configurations react in different ways to the adding of new devices: while the waterfall sharing scheme configuration has worsen its performances (from an average data rate of 1400 - 1800 kbps to one of about 1100 kbps), in fact often the star sharing scheme has increased its performances since it has passed from an average data rate of 1200 - 1800 kbps to one of 1600 - 2000 kbps. This fact is concordant with the results of mobiles: also on Android platform the star sharing scheme coupled with WiFi-Direct works better than the waterfall sharing scheme with the same wireless protocol. Moreover the average data rate of almost 2000 kbps represents the first time that a desktop configuration behaves better than its mobile correspondent.

8.2.2.3 Standard deviation of data rate comparison

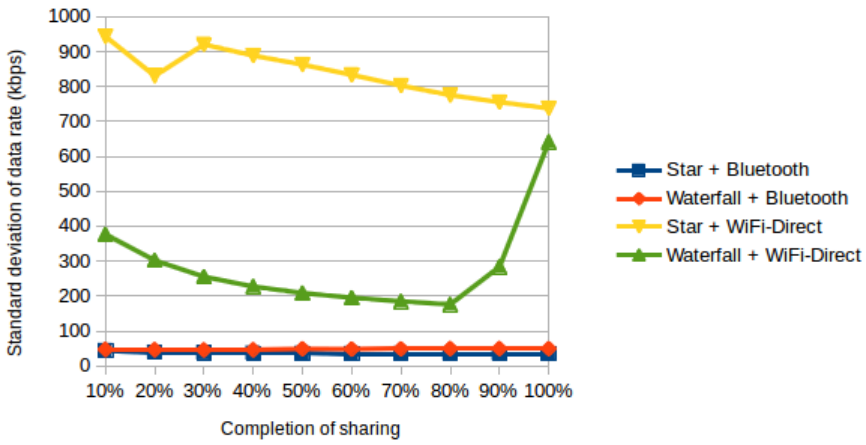


Figure 8.16: Standard deviation of data rate comparison of the configurations with 5 devices (desktop)

From the Figure 8.16 it is clear that the two Bluetooth configurations are very stable: they keep a standard deviation of more or less 50 kbps all the sharing long. The ratio between the standard deviation and the average data rate is quite normal, about $\frac{1}{5}$. However, looking at the results with 3 devices, it appears that while the star sharing scheme has kept more or less the same ratio, the waterfall sharing scheme has worsen its performances since the ratio has increased. Comparing the results with the mobile corresponding ones it seems that the configurations tends to be similar: the mobile ones just has a lightly higher standard deviation.

The most particular configurations are the WiFi-Direct ones: the star sharing scheme configuration starts with a very high standard deviation (almost 1000 kbps)

which is gradually reduced till 700 kbps, while the waterfall sharing scheme configuration starts with a standard deviation of 400 kbps, which arrives to 200 kbps and then increases till 600 kbps. It is important to note that the star sharing scheme configuration traces again the "wave" behaviour which is present in the scenario with 3 devices, while the waterfall sharing scheme does not. The ratios between standard deviation and average data rate are not particularly good: both of them record a ratio of about $\frac{1}{2}$, but it highly vary since both average data rate and standard deviation also vary.

8.2.3 Considerations on desktop experiments

Considering the Bluetooth configurations, the star sharing scheme configuration traces a better behaviour than the waterfall sharing scheme, especially considering 3 devices only: in fact, the adding of new devices leads the two configurations to be very similar from the performances point of view. The fact that the two configurations have the same results when the number of devices is increased means that the star sharing scheme coupled with Bluetooth does not born well the adding of new devices, while in waterfall sharing scheme configuration the degradation of the performances is minor. This evidence is concordant with the results presented of mobile environment, in which the waterfall sharing scheme coupled with Bluetooth works better than the other one with the same wireless protocol once the number of devices increases: the two platforms tend to have similar results considering the Bluetooth configurations, even if the mobile configurations record sightly better results, especially from the total time involved point of view.

Looking at the WiFi-Direct configurations, the star sharing scheme configuration keeps to seem the best solution once the number of devices increases, just as in mobile environment. However the very high standard deviation shown by the desktop configuration signals a very hiccups transmission: from this point of view, the mobile solution works better. Is important to note that the same considerations which have been made on the cause of the high standard deviation in mobile environment are valid also for the desktop environment: since WiFi-Direct is a high-speed protocol, the thread which uses it to transmit data towards other devices has often to wait for new data brought by the other thread. Anyway, the same behaviour has been observed also in some of the devices which acted as transmitters or first nodes from the output connection point of view, even if a lot of data were available to be transmitted (in some cases the download of the whole resource has been really quick, especially in desktop environment): this means that WiFi-Direct tends to trace the bursty TCP connections, both for lack of data to be sent and for design of the protocol. Just as in mobile environment, the configurations tend to trace very similar results comparing total time involved, and also comparing the results with mobile platform ones the performances are almost equal: so the configurations behave very similar independently from the platform, despite the differences on average data rate and standard deviation. The fact that the total time involved is more or less the same both for stable and hiccups transmissions means that hiccups transmissions are not necessarily worse than the other ones: they compensate the high oscillations with a very high data rate reached in certain moments during the sharing.

8.3 Final results

The more important results of the data collection can be summarized in few points:

- from the point of view of data rate and total time involved, WiFi-Direct is surely better than Bluetooth;
- from the energetic point of view, WiFi-Direct represents a better solution than Bluetooth, even if Bluetooth is better when the device has only to receive data from a source;
- the waterfall sharing scheme is more suitable for the adding of new devices when Bluetooth is used as wireless protocol;
- the star sharing scheme is more suitable for the adding of new devices when WiFi-Direct is used as wireless protocol;
- Bluetooth has shown its best results on mobile platform, especially from the time involved point of view;
- WiFi-Direct has shown its best results on mobile platform, especially from the point of view of standard deviation - average data rate ratio;
- considering the number of errors encountered during the sharing, the desktop solution is certainly better than the mobile one.

These considerations lead to consider the star sharing scheme coupled WiFi-Direct as the best solution for the server-client problem described in Chapter 1 since it allows an almost instantaneous propagation of the file downloaded with the best trade-off between data rate and energy consumptions. The Bluetooth solutions can be used in scenarios in which a high interaction with other devices is needed, but the download of a file is not requested, like offline multi-player games: in these cases the energy saving properties of Bluetooth can be exploited and since generally it is not needed to share big resources, Bluetooth data rate is suitable for the needs of this use case.

Chapter 9

Conclusions

The chapter shows the consequences of the work performed and the aspects that have been evaluated at the end of the project. In particular, a comparison between the estimation on the time involved and the actual expenditure is presented, than present and future works linked to the project are described.

9.1 Estimation - final comparison

The specifications of the project contain an estimation of the time involved to complete the work: in particular, each activity of the project has its own estimation. The cumulative estimation consists in 310 hours of work while the project has asked 315 hours at present since some activities have been underestimated: these are especially the tests writing and the data collection, which have shown the described problems whose solutions were not immediate. Table 9.1 shows the comparison between the estimates and the number of hours involved at present.

Activity	**Expected**	**Actual**	**Difference**
A1 - Study of the packages for Bluetooth and WiFi-Direct communication	30	24	-6
A2 - Study on the integration between framework, graphic interfaces and communication packages	20	17	-3
A3 - Analysis	20	19	-1
A4 - Design	40	36	-4
A5 - Implementation	125	120	-5
A6 - Verification and tests	40	48	+8
A7 - Data collection	15	25	+10
A8 - Documentation	20	26	+6

Table 9.1: Estimation - final comparison

As the table shows, some hours of the preliminary study before starting the development have been saved since the availability of guides on the communication packages has lighten the activity. Moreover the well documented frameworks and libraries have helped to save further hours in implementation, while some hours have been saved from

design thanks to the fact that a project which used one of the two frameworks (Qt) has been already developed during the software engineering teaching so the framework way of working was quite known. It is important to note that also the writing of the documentation has asked more time than the estimated one since the description of the way of working of the software had to be clear but specific at the same time in order to allow any maintainer to understand the structure of the application without problems and it had to be doubled since Android and desktop versions have a lot of common aspects but they are not equal: moreover the writing in English language has made heavier the work. The chart in Figure 9.1 shows the results described in a clearer way.

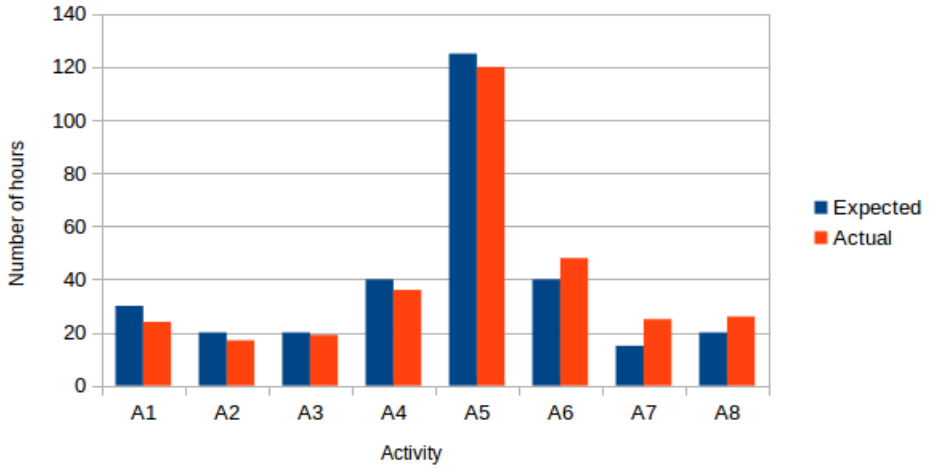


Figure 9.1: Estimation - final comparison chart

9.2 Workshop participation

The results shown in Chapter 8 have been partially used to write an article which has been submitted to the 11^{th} International Workshop on Networking Issues in Multimedia Entertainment (NIME'15): the conference will take place on February 2015, in Anaheim, California. The article , whose name is "BlueFall: Testing Swarming Protocols through Mobile Phones", has been written with the cooperation of Prof. Claudio Enrico Palazzi and Armir Bujari, PhD researcher of University of Padua. In the article have been included the results gained on Android platform as the issues encountered and the solutions adopted during the development of the application. The article can be found in appendix A.

9.3 Final considerations

BlueFall project leaves open a lot of doors for further developments: first of all, BlueFall has been developed for Android platform, but there are many other mobile platforms that may be used in order to individuate any improvement or worsening of the sharing scheme proposed: however, the motivation for any possible difference is

expected to be the different WiFi and Bluetooth devices rather than the different operating system. Then, a lot of other wireless protocols could be tested and other swarming protocols could be implemented. Future works include the development of BlueFall on iOS and Windows Phone.

The most important activities of the stage have been the data collection and the analysis of the results because they correspond to a real research work: the data collection has shown some problems that were almost unknown before these experimentations and it has allowed the usage of tools designed specifically for research activities, like PowerMonitor. Instead the analysis of the results has allowed to bring new important results on a technology which is not yet famous, like WiFi-Direct. So the BlueFall project is really interesting because the motivations which have led to its development are extremely current. Moreover, the fact that it has been designed with a flexible structure is indicative of the fact that the project is far from being closed: further works may include tests of the sharing schemes coupled with other wireless protocols, like NFC or infrared, for example. Another exciting feature is the fact that the devices used in desktop applications are changeable, so new tests may be performed using Bluetooth 4.0 or newest WiFi devices and none of these tests asks to change the code of the application. In few words, the stage activity have allowed to create a research tool which can keep to bring interesting results for a long time and which is usable in other researches with few modifications: this one is probably the most important and rewarding aspect of the project.

Appendix A

Workshop paper

The BlueFall project has led to the drawing up of a research article which has been submitted to the the 11^{th} International Workshop on Networking Issues in Multimedia Entertainment (NIME'15). The contents of the work are reported here.

BlueFall: Testing Swarming Protocols through Mobile Phones

Matteo Pozza
Università degli Studi di Padova
Email: matteo.pozza@studenti.unipd.it

Claudio Enrico Palazzi
Università degli Studi di Padova
Email: cpalazzi@math.unipd.it

Armir Bujari
Università degli Studi di Padova
Email: abujari@math.unipd.it

***Abstract*—In recent years, the number of appliances connected to the Interned has steadily increased thanks to the wide popularity of WiFi/3G enabled mobile devices. This has led to an increase of multimedia resource consumption, causing traffic overload problems for classic resource providers. Different solutions have been studied to shift this traffic overhead from the server(s) towards the client(s) through the employment of the so called *swarming protocols*. However, most of these solutions have been verified only through simulations. While simulations consist in a necessary first step, they can be far from the actual outcome of employing real protocols and devices. To this end, in this paper we describe *BlueFall*, an application that could allow to test swarming protocols using actual mobile phones. Pursuing our goal, we provide some preliminary evidence on two simple swarming schemes we have implemented in our testbed.**

I. Introduction

Since the market launch of the first Internet enabled mobile phone, the number of devices connected to the Internet has continued to grow. In addition, high speed Internet access technology has become a standard for mobile devices (e.g., WiFi or 3G/4G) allowing the consumption of many and fat resources from the Internet, such as rich multimedia content. In this context, the classic client-server paradigm starts to waver since both the number of consumers and the weight of the resources keep increasing, leading to critic infrastructure-side stress and/or consumer throughput degradation. One solution to the problem is the mirroring of the server(s), but this is an expensive solution which only postpones the problem and does not tackle the core issue.

To this end, swarming protocols have been proposed which have a common *modus operandi*, that of shifting the burden of operations from the server to the client side. Simply put, if several clients request the same content, the server can provide the resource to just one of them and let the clients share the resource among themselves [1]–[3]. Alternatively, single pieces of the considered resource can be retrieved by all the clients in different moments in time and chunks can later be exchanged among them, like in [4], [5]. Through these protocols, clients are no more the end-point of connections with the servers, but they can become servers-offshoots for other clients.

Unfortunately, swarming solutions proposed so far have been generally validated through simulation studies [6]–[8]. Although the simulation environment does offer some advantages, e.g., by allowing the complete control of the variables involved in the experiment, they are inherently limited and results can differ substantially from real field trials. To this end, reducing the gap between theoretical results and real ones we set on a trial to implement and validate a testbed application, named BlueFall. Our proposal allows to fetch a resource from the Internet and successively makes it available to others by means of two different swarming protocols. The targeted development platform is Android-based and the swarming protocols make use of either Bluetooth or WiFi-Direct for resource sharing. In the following sections we discuss the testbed design and its implementation, the issues we encountered and the solutions proposed to address them. Furthermore, we provide a performance evaluation of the swarming schemes.

II. Related works

Nandan *et al.* in [4] and Fiore *et al.* in [10] propose and discuss different protocols and algorithms tailored for communications in vehicular ad hoc networks (VANETs). In both cases, the validation is performed through simulation studies. In this context, our platform can be adopted to fully replace the simulated short-range wireless transmissions, hence to gain a realistic outcome of the protocols performance.

In [9] a VANET protocol is presented and validated through simulations. The authors study their proposal by taking into consideration the node density problem, validating their solution with a varying node population. Our BlueFall could be adopted in this scenario by employing a certain number of mobile phones placed at appropriate distances. While the number of mobile devices is limited, they could suffice to help establish a ground truth for low node density scenarios.

Finally, Yasumoto *et al.* present a study focused on energy saving analysis for mobile devices employing short-range communication protocols, like Bluetooth and WiFi [12]. Yet again, by employing BlueFall and other appropriate tools allowing precise battery measurement, being those hardware [13] or software [14], one could complement the analysis with field measurements. Concluding, while simulations are a necessary first step to validate an idea we argue that they are not a sufficient one. Our proposed testbed can be a trade off solution which could enable one to gain realistic results at restrained costs considering the popularity of Android devices supporting both Bluetooth and WiFi technologies.

In the following section we discuss our testbed environment, the entities it is composed by and the protocol *modus operandi*. For the sake of clarity, we point out that the term protocol or sharing scheme is used interchangeably and both are used to denote the type of swarming technique being employed.

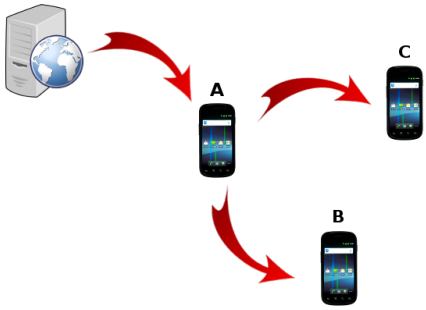


Fig. 1: The *"star" sharing scheme*.

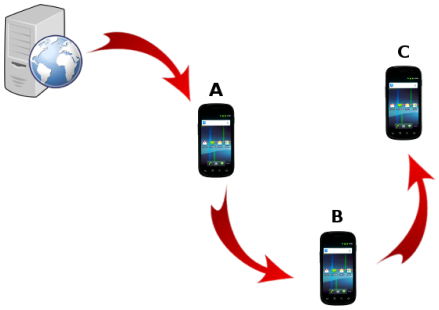


Fig. 2: The *"waterfall" sharing scheme*.

III. The Envisioned Scenario

Our testbed models an environment composed of a certain set of elements: a resource, or a number of resources retrieved sequentially by n nodes ($n \quad 0$), and a resource provider, acting as a server accessible through an Internet connection. The resource is directly retrieved only by a single device and this one in turn immediately makes it available for others and the modality depends on the adopted swarming scheme. We assume that every device is able to communicate using either Bluetooth or WiFi-Direct: these technologies have been chosen since most of the recent Android devices are equipped with both of them. The connections to the server are reduced to one, and the server is not aware of the existence of any other consumer *n-1* device involved in the network. Hence, the resource can be retrieved from the server using a single WiFi or 3G/4G connection. In specifics, the proposed sharing schemes are:

> *"Star" sharing scheme*: in this scenario, device (A) retrieves the resource from the server directly and immediately shares it with the other devices (B and C) using either Bluetooth or WiFi-Direct. We call this scenario a *centralized* sharing scheme since device A acts as a server for the other two devices (Figure 1);
>
> *"Waterfall" sharing scheme*: in this scenario, device (A) retrieves the resource from the server directly and immediately shares it with another device (B), which in turn shares it with another device (C). In this scenario, device (B) is contemporarily retrieving the remainder of the content from the initial device (A) and also fetching the content to device (C). This represents a *distributed* sharing scheme since every device (except the last one in the chain) acts as a resource gateway for another device (Figure 2) [15].

IV. The Application

The aim of our devised testbed and deployment scenarios is to offer the possibility to collect real data related to the network performance. Considering the above scenarios, one could obtain measurements on both the incoming data stream (the data in-flow originating from Internet or a point-to-point connection with another device) and the outgoing data stream (the data out-flow from the servant device). The measurements could include the instantaneous, average data rate and the total time involved in resource retrieval. All of these metrics are made available to the end-user through an appropriate view on the device screen involved in the sharing process.

In addition to real time statistics regarding the flows, a log file of the entire operations can be generated containing the information related to the resource *per se*, and the transmission trend in time. This trend is computed by sampling the flows in time and the software provides a dedicated view where this feature could be customized (e.g., by configuring the number of samples and the log file format).

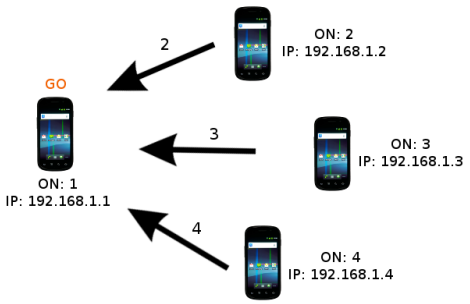
A. Design and Implementation

All the studied scenarios and relative configurations start with an initial node requesting a content and immediately sharing it with the rest of the network nodes. Our software handles the combined retrieval and sharing process by decoupling the two activities allowing them to work on different regimes on a single common buffer. This approach follows the producer-consumer design pattern, where the producer corresponds to the task of retrieving the initial content and the producer to the task of sharing the resource with the other devices. However, another design choice emerges concerning the buffer size. Two types of solutions could be employed:

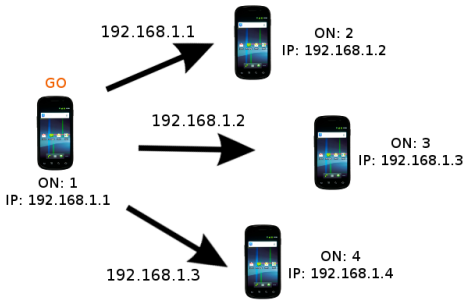
> allocate a buffer as large as the resource being shared;
>
> allocate a buffer of smaller dimensions compared to the resource size.

Both solutions present pros and cons. The first approach allows to reduce the software complexity and speeds up the overall process as the producer task never stops its activity. Indeed, in this scenario the buffer is not full until the resource has been entirely retrieved and when this happens, its work is finished. However, the main problem with this approach is that the size of the resource being retrieved is limited to the amount of RAM the application is allowed to use. The second approach on the other side, if devised properly, could alleviate the RAM problem but comes at a higher cost in terms of in-application memory management. In the current version, we decided to follow the first approach but the modularity of the application allows to easily add another memory management technique. Moreover, given the scope of our application, we are more interested in profiling the communication costs rather than local operations which are platform and vendor dependent.

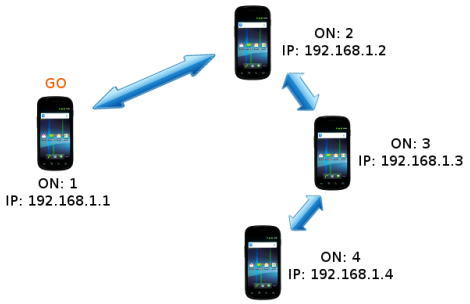
Another problem we had to face regards the WiFi-Direct set up. In [11], the authors argue that different ways exist in



(a) The devices communicate the established order to the group owner.



(b) The group owner communicates the neighbor IP address to each of the devices.



(c) The waterfall scenario configuration is complete.

Fig. 3: The *waterfall scenario set up* phases.

order to create a WiFi-Direct network, namely: the standard, autonomous and persistent modes. In this regard, we have adopted the autonomous mode since it is the simplest among all and it also avoids the GO-negotiation phase. It is important to point out that an Android device cannot belong to more than one WiFi-Direct network at the same time. While this is not a problem when we adopt a "star" configuration, every device joins the group formed by the group owner of the network, in the "waterfall" configuration this becomes a big constraint since a single device has to connect to two devices. We could workaround this problem by allowing only one group formation between all the involved devices but this leads to additional complexity as every device knows only its own IP address and that of the group owner. In other words, the devices comprising the network cannot communicate among them, except with the group owner. The group owner instead knows all the IP addresses of the devices in the network but it does not know which device needs which IP address. We address this problem by introducing a *waterfall set up* phase consisting of the following steps:

1) all the devices involved in the waterfall (group owner excluded) communicate their order number (ON), a number that represents their position in the waterfall configuration, to the group owner which in turn creates an hash table with the IP addresses as keys and the order number as a value (Figure 3a);
2) the group owner communicates to every device the IP address of the device that has an order number immediately preceding its one (Figure 3b);
3) all the devices know the IP address of the device preceding them and a connection set up phase can initiate (Figure 3c).

Yet, another issue we had to tackle while developing our testbed presented itself in the scenario of multiple resource sharing in the "waterfall" configuration. The symptom presented between the first (initiator) device and the last one, but in the general case it could occur between the first device and the others in the chain which are working on different regimes. This erroneous state occurred when relatively low transmission rate protocols like Bluetooth were employed giving spur to a situation where the first device had finished retrieving one resource and successively goes to retrieve and share the next one, while the last (or middle) device is still retrieving and/or sharing the previous one. If not dealt accordingly this could lead the other nodes to naively interpret this new data as parts of the old one resulting in an overall data corruption.

We tackled this issue by introducing a feedback mechanism whereby nodes communicate to their parents in the chain to go and feed the next available resource (Figure 4a and 4b). In specifics the feedback mechanism is comprised the following operations occurring independently in each node:

the first node of the waterfall only listens for a feedback;

an intermediate node first listens for a feedback, and once received it sends a feedback;

the last node of the waterfall only sends a feedback.

V. Simulation Testbed and Configuration

We took particular care in testing our platform with emphasis on the proposed sharing schemes. The testbed consists of three nodes, one retrieving the resource and the other two acquiring it through either Bluetooth or WiFi-Direct. In the experiments we used Samsung GT-i9250 devices with Bluetooth protocol ver. 3.0. The resource being retrieved is a *.deb* package with a size of 10 MBytes retrieved by the first node using a WiFi Internet connection. In our experimentation, we decided to test 4 different configurations:

Star configuration + Bluetooth technology;

Star configuration + WiFi-Direct technology;

Waterfall configuration + Bluetooth technology;

Waterfall configuration + WiFi-Direct technology.

(a) The order in which the devices listen for the feedback.

(b) The order in which the devices send the feedback.

Fig. 4: The *feedback mechanism*.

In particular, we run 20 trials for each configuration. The results are obtained by averaging all the acquired values. The actual configuration uses 10 sampling points per log file taken at regular intervals: the first sample is taken once 10% of the whole resource has been gained, the second one once 20% of the whole resource has been gained, and so on.

In our experiments, we profile the in-flows at the receiving devices, that is the devices which only receive the data content and are not engaged as servers for other devices. In the "star" configuration this corresponds in measurements for all the *n-1* devices receiving the content, while in the "waterfall" configuration we profile only the last device of the chain. Regarding the battery consumption evaluations, we employ an hardware tool [13] that allows very precise measurements by in-place substitution of the device battery. In particular, we identify four energetic profiles to be evaluated:

the *transmitter*, that is the node that, in the "star" sharing scheme, retrieves the resource from Internet and transmits it to all the other nodes;

the *first node*, that is the node that, in the "waterfall" sharing scheme, retrieves the resource from Internet and sends it to the second node;

the *intermediate node*, that is the node that, in the "waterfall" sharing scheme, retrieves the resource from a node and sends it to another one;

the *receiver* (or *last node*), that is a node that only receives the resource from another node and it doesn't transmit the resource to anyone.

The battery data collection consists on the average power consumption recorded at the end of several sharing operations. To not bias the measurements, the device screen has been set with a maximum brightness and all the device sensors (e.g., 3G/4G, NFC) have been switched off. The baseline value for screening operations is computed by having the device on the main screen for about 10 minutes using a custom created background image [1] and switching off all the device sensors.

[1] solid color, HUE: 0, SATURATION: 5, VALUE: 33, RED: 85, GREEN: 81, BLUE: 81

VI. Results

In this section, we provide and discuss the performance metrics for each of the devised configurations. We study the overall time involved in retrieving the resource, the average data rate and the battery consumption related to the resource retrieval only.

From the chart in Figure 5, it appears clear that, since Bluetooth cannot compete with WiFi-Direct on data transmission speed, the configurations employing WiFi-Direct require less time for retrieval than the ones employing Bluetooth. However, it is interesting to note the comparison between the two Bluetooth configurations: the "star" scenario requires less time to complete than the "waterfall" one, while in the WiFi-Direct configuration the opposite occurs. This difference in outcome in the Bluetooth scenario is to be attributed to the nature of the overlay network being formed and slot attribution process which in the "star" scheme corresponds to a piconet and in the "waterfall" results in a scatternet formation.

Concluding, WiFi-Direct and Bluetooth configurations have an average throughput of about 2 Mbps and 700 Kbps, respectively.

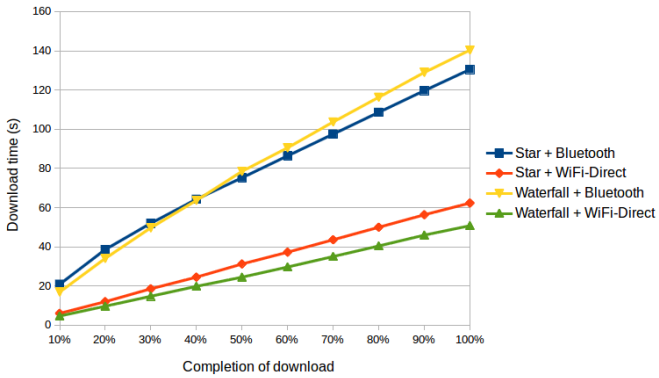


Fig. 5: Time comparison between the different configurations.

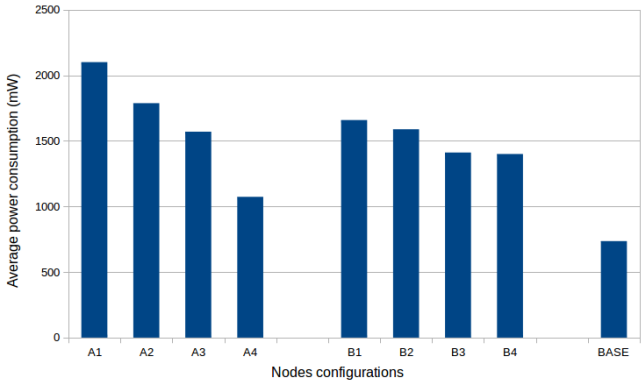


Fig. 6: Energy consumption comparison between the different configurations. Group A indicates the Bluetooth roles, while the group B indicates the WiFi-Direct ones. The roles are: 1 - transmitter, 2 - first node, 3 - intermediate node, 4 - receiver. BASE stands for baseline value.

Concerning the energy consumption, one expects WiFi-Direct to consume a lot more energy when compared to Bluetooth. However, this is only partially true. Looking at the chart in Figure 6 one can notice that "passive" nodes, those only receiving the content, in a WiFi-Direct setting have a higher energy consumption when compared to Bluetooth ones. On the contrary, in all the remaining roles the Bluetooth devices consume more than the WiFi-Direct ones.

Another observation is that while Bluetooth roles exhibit a higher degree of heterogeneity among themselves in terms of energy consumption, the WiFi-Direct ones are more homogeneous. This implies that the energy consumption in the WiFi-Direct scheme is independent from the role that the device assumes. This is particularly evident looking at the comparison between the intermediate device and the receiver one(s): while in Bluetooth roles there is a difference of about 500 mW, in WiFi-Direct the difference is less than 100 mW.

A plausible explanation for these higher energy profiles of Bluetooth, is that this technology is not suited for large data transfers as in our case. In fact, Bluetooth was designed to be a cable replacement protocol for short and discontinuous data transmissions. Instead, in our testbed, Bluetooth is used very intensively and for a continuous period of time. This implies the inhibition of the energy-saving property of the protocol. Moreover, in some of the roles, Bluetooth works along with WiFi at the same time since the initiator device also is involved in retrieving the resource from Internet. Furthermore, taking into consideration the error prone nature of wireless transmissions in the Bluetooth scenario, potential retransmissions lead to a higher energy consumption. The low energy consumption of the receiver role is due to the fact that the node only receives the resource and any energy consumption due to retransmissions is charged to the node that is sending the resource.

Looking at the WiFi-Direct roles, the small differences between the energy consumptions are due to the fact that WiFi-Direct is designed to be a robust protocol for high speed data transmissions, and this is exactly the usage it was employed for in our testbed. Moreover, in order to retrieve the resource(s) from Internet and to successively share it with others, the same device uses also the WiFi interface. This avoids further interferences and does not require any additional device component switched on (like in the corresponding Bluetooth roles), leading to a further energy saving.

VII. Conclusion and Future Work

In this paper we proposed a testbed that exploits mobile phones to gain realistic measures about swarming protocols. We implemented two simple swarming protocol than can work as bases for more complex ones. We tested our proposal employing two short-range transmission technologies available from our mobile devices which are Bluetooth and WiFi-Direct.

The obtained results show that WiFi-Direct and a sharing scheme in couple can be a valid method to reduce the work that is requested to a single resource provider in a classic server-client mode with many clients downloading multimedia files.

References

[1] G. Marfia, M. Roccetti, A. Amoroso, G. Pau, "Safe Driving in LA: Report from the Greatest Intervehicular Accident Detection Test Ever", IEEE Transactions on Vehicular Technology, IEEE Vehicular Technology Society, 62(2), Feb 2013.

[2] W. Ben Jaballah, M. Conti, M. Mosbah, C. E. Palazzi, "A secure alert messaging system for safe driving", Computer Communications 46, Jun 2014.

[3] W. Ben Jaballah, M. Conti, M. Mosbah, C. E. Palazzi, "Fast and Secure Multihop Broadcast Solutions for Intervehicular Communication", IEEE Transactions on Intelligent Transportation Systems 15(1), Feb 2014.

[4] A. Nandan, S. Das, G. Pau, M.Y. Sanadidi, M. Gerla, "Cooperative Downloading in Vehicular Ad Hoc Wireless Networks", in Proc. of IEEE/IFIP International Conference on Wireless On demand Network Systems and Services (WONS '05), St. Moritz, Switzerland, Jan 2005.

[5] D. Maggiorini, C. Quadri, L.A. Ripamonti, "On the Feasibility of Opportunistic Collaborative Mixed Reality Games in a Real Urban Scenario", in Proc. of the International Conference on Computer Communications and Networks, (ICCCN 2012), Munchen, Germany, Jul 2012.

[6] M. Gerla, D. Maggiorini, C. E. Palazzi, A. Bujari, "A Survey on Interactive Games over Mobile Networks", Wireless Communications and Mobile Computing 13(3), Feb 2013.

[7] C. E. Palazzi, A. Bujari, "Social-Aware Delay Tolerant Networking for Mobile-to-Mobile File Sharing", International Journal of Communication Systems 25(10), Oct 2012.

[8] C. E. Palazzi, A. Bujari, G. Marfia, M. Roccetti, "An Overview of Opportunistic Ad Hoc Communication in Urban Scenarios", in Proc. of the 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET 2014), Piran, Slovenia, Jun 2014.

[9] U. Lee, J.S. Park, J. Yeh, G. Pau, M. Gerla, "Code Torrent: Content Distribution Using Network Coding in VANET", in Proc. of the International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (MobiShare '06), Los Angeles, CA, USA, Sep 2006.

[10] M. Fiore, J.M. Barcelo-Ordinas, "Cooperative download in urban vehicular networks", in Proc. of the IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS '09), Macau (S.A.R.), China, Oct 2009.

[11] D. Camps-Mur, A. Garcia-Saavedra, P. Serrano, "Device to device communications with WiFi Direct: overview and experimentation", IEEE Wireless Communications Magazine, 20(3), Jun 2013.

[12] K. Yasumoto, Y. Takamatsu, W. Sun, M. Ito, "Energy-Efficient Cooperative Download for Smartphone Users through Contact Time Estimation", in Proc. of the IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob '12), Barcelona, Spain, Oct 2012.

[13] Monsoon Power Monitor. https://www.msoon.com/LabEquipment/PowerMonitor. Last visited: 30 June, 2014.

[14] PowerTutor - A Power Monitor for Android-Based Mobile Platforms. Available at: http://ziyang.eecs.umich.edu/projects/powertutor/. Last visited 10 June 2014.

[15] A. Hamidian, C. E. Palazzi, T. Y. Chong, J. M. Navarro, U. Korner, M. Gerla, "Deployment and Evaluation of a Wireless Mesh Network", in Proc. of the 2nd International Conference on Advances in Mesh Networks (MESH 2009), Athens, Greece, Jun 2009.

Bibliography

[1] A. Nandan, S. Das, G. Pau, M.Y. Sanadidi, M. Gerla, "Cooperative Downloading in Vehicular Ad Hoc Wireless Networks", in Proc. of IEEE/IFIP International Conference on Wireless On demand Network Systems and Services (WONS '05), St. Moritz, Switzerland, Jan 2005.

[2] U. Lee, J.S. Park, J. Yeh, G. Pau, M. Gerla, "Code Torrent: Content Distribution Using Network Coding in VANET", in Proc. of the International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (MobiShare '06), Los Angeles, CA, USA, Sep 2006.

[3] M. Fiore, J.M. Barcelo-Ordinas, "Cooperative download in urban vehicular networks", in Proc. of the IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS '09), Macau (S.A.R.), China, Oct 2009.

[4] D. Camps-Mur, A. Garcia-Saavedra, P. Serrano, "Device to device communications with WiFi Direct: overview and experimentation", IEEE Wireless Communications Magazine, 20(3), Jun 2013.

[5] K. Yasumoto, Y. Takamatsu, W. Sun, M. Ito, "Energy-Efficient Cooperative Download for Smartphone Users through Contact Time Estimation", in Proc. of the IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob '12), Barcelona, Spain, Oct 2012.

[6] M. Paone, L. Paladina, D. Bruneo, A. Puliafito, "A Swarm-based Routing Protocol for Wireless Sensor Networks", in Proc. of the IEEE 6th International Symposium on Network Computing and Applications (NCA '07), Cambridge, Massachusetts, USA, Jul 2007.

[7] Slides of the Wireless Networks course taught by Claudio Enrico Palazzi. Academic year 2013/2014. Available at: http://www.math.unipd.it/~cpalazzi/retiwireless.html.

[8] Slides of the Software Engineering Module A taught by Tullio Vardanega and Riccardo Cardin. Academic year 2013/2014. Available at: http://www.math.unipd.it/~tullio/IS-1/2013/.

[9] Slides of the Software Engineering Module B taught by Tullio Vardanega and Riccardo Cardin. Academic year 2013/2014. Available at: http://www.math.unipd.it/~rcardin/sweb.html.

[10] The official site of the Android framework: http://developer.android.com/index.html.

[11] The official site of the Qt framework: http://qt-project.org/.

[12] Java Excel API - A Java API to read, write, and modify Excel spreadsheets. Available at: http://jexcelapi.sourceforge.net/.

[13] BlueZ - Official Linux Bluetooth protocol stack. Available at: http://www.bluez.org/.

[14] Linux WPA/WPA2/IEEE 802.1X Supplicant. Available at: http://w1.fi/wpa_supplicant/.

[15] LibXL - excel library for developers. Available at: http://www.libxl.com/download.html.

[16] RapidXml library official page: http://rapidxml.sourceforge.net/.

[17] cURL library official page: http://curl.haxx.se/.

[18] The Signal-Slot mechanism documentation: http://qt-project.org/doc/qt-5/signalsandslots.html.

[19] Mockito - simpler & better mocking. Available at: https://code.google.com/p/mockito/.

[20] PowerMock - a Java framework that allows you to unit test code normally regarded as untestable. Available at: https://code.google.com/p/powermock/.

[21] QTestLib manual: http://qt-project.org/doc/qt-4.8/qtestlib-manual.html.

[22] CppUTest - unit testing and mocking framework for C/C++. Available at: http://cpputest.github.io/.

[23] Eclipse official page: https://www.eclipse.org/.

[24] QtCreator official page: http://qt-project.org/wiki/category:tools::qtcreator.

[25] Javadoc tool official page: http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html.

[26] Doxygen official page: http://www.stack.nl/~dimitri/doxygen/.

[27] List of hardware devices which works with ath9k driver: http://wireless.kernel.org/en/users/Drivers/ath9k/products.

[28] List of hardware devices which works with carl9170 driver: http://wireless.kernel.org/en/users/Drivers/carl9170#available_devices.

[29] PowerTutor - A Power Monitor for Android-Based Mobile Platforms. Available at: http://ziyang.eecs.umich.edu/projects/powertutor/. Last visited 10 June 2014.

[30] Monsoon Power Monitor. https://www.msoon.com/LabEquipment/PowerMonitor. Last visited: 30 June, 2014.

Acronyms

API Application Programming Interface. 14, 28

DHCP Dynamic Host Configuration Protocol. 32

GBps gigabytes per second. 18

Gbps gigabits per second. 18

GUI Graphical User Interface. 23, 44

HSDPA High Speed Downlink Packet Access. 1, 56

HTML HyperText Markup Language. 56

HTTP HyperText Transfer Protocol. 28

ICS Ice Cream Sandwich. 14

JVM Java Virtual Machine. 41, 43

kBps kilobytes per second. 18

kbps kilobits per second. 18, 63, 65, 66, 72, 73, 76, 77

LTS Long Time Support. 15

MBps megabytes per second. 18

Mbps megabits per second. 18

MVC Model View Controller. 23–26, 28

NFC Near Field Communication. 56, 81

P2P Peer to peer. 67

RAM Random Access Memory. 41, 44

TCP Transmission Control Protocol. 77

UML Unified Modeling Language. 19

USB Universal Serial Bus. 51, 52, 59

WYSIWYG What You See Is What You Get. 44

XML eXtensible Markup Language. 18, 24–27, 30–32