OMAR EDUARDO APONTE QUERALES

MOBILITY MANAGEMENT OPTIMIZATION VIA INFERENCE OF ROAMING BEHAVIOR

Orientadora: Prof. Dr. Helena Rute Esteves Carvalho Sofia

Professora Associada/Investigadora Sénior

Universidade Lusófona de Humanidades e Tecnologias

Escola de Comunicação, Arquitetura, Artes e Tecnologias de Informação

COPELABS - Unidade de Investigação em Computação Cognitiva e Centrada nas Pessoas

Lisboa

2019

OMAR EDUARDO APONTE QUERALES

MOBILITY MANAGEMENT OPTIMIZATION VIA INFERENCE OF ROAMING BEHAVIOR

Tese defendida em provas públicas na Universidade Lusófona de Humanidades e Tecnologias no dia 4 de Julho de 2019, perante o júri, nomeado pelo Despacho de Nomeação n. 165/2019, de 5 de Junho, com a seguinte composição:

Presidente: Prof. Doutor José Luis Azevedo Quintino Rogado (ULHT)
Arguente: Prof. Doutor José Carlos Ferreira (ISCTE-IUL)
Vogal: Prof. Doutor Pedro Alexandre Reis Sá da Costa (ULHT)
Orientadora: Prof. Doutora. Helena Rute Esteves Carvalho Sofia (ULHT)

Universidade Lusófona de Humanidades e Tecnologias

Escola de Comunicação, Arquitetura, Artes e Tecnologias de Informação

COPELABS - Unidade de Investigação em Computação Cognitiva e Centrada nas Pessoas

Lisboa

2019

Abstract

Movement estimation techniques have been applied for long in wireless and cellular networks with the aim to provide better support for networking operational aspects, such as resource management while devices are on the move. For instance, techniques for fast handover based on movement anticipation have been a topic extensively addressed, e.g., within the context of the *Internet Engineering Task Force (IETF)* [1, 2]. Such techniques have been often explored statistically, based on data earlier collected from, for instance, cellular customers. Hence, mobility estimation has been mostly applied from an operator perspective.

In the most recent years techniques such as virtualization and predictive analysis bring in the possibility to explore mobility estimation from an end-user perspective as well. Mobility estimation applied from the end-user perspective is relevant, as it allows for a finer grained detail of roaming behavior and thus provides the means to better understand user movement patterns, both from an individual and a collective perspective.

Being capable of anticipating movement is relevant to optimize the network operation, be it from a mobility management perspective (e.g., handover optimization), from a resource management perspective (e.g., performing a more intelligent load-balancing), or from a routing viewpoint (e.g., making routing more stable by selecting paths that have a chance to be more stable in variable topologies).

This dissertation contributes to the topic of applicability of mobility estimation in the context of mobility management, via: i) analysis and proposal of mobility estimation functions; ii) integration of the developed utility functions into an existing software application (NSense); iii) validate the different functions based on realistic settings (testbed).

Keywords: mobility tracking; social mobility behavior; user-centric networks.

Resumo

As técnicas de estimativa de movimento têm vindo a ser aplicadas em redes sem fios e redes celulares, com o objetivo de fornecer melhor suporte nos aspectos operacionais das redes, como gerenciamento de recursos enquanto os dispositivos estão em movimento. Por exemplo, técnicas para “fast handover” com base na antecipação de movimento tem são um tópico relevante em gestão de mobilidade. Por exemplo, na *Internet Engineering Task Force (IETF)* [1, 2] tais técnicas têm sido frequentemente exploradas estatisticamente, com base em dados coletados de clientes de redes celulares. Assim, a estimativa de mobilidade tem sido aplicada principalmente a partir de uma perspectiva do operador.

Nos anos mais recentes, técnicas como virtualização e análise preditiva trazem a possibilidade de explorar a mobilidade a partir de uma perspectiva do usuário final. A estimativa de mobilidade aplicada da perspectiva do usuário final é relevante, pois permite um detalhe mais refinado do comportamento de “roaming” e, assim, possibilita a detecção de padrões de movimento do usuário, tanto de uma perspectiva individual quanto de uma perspectiva coletiva.

A possibilidade de se prever movimento de uma perspectiva de rede é relevante quer de uma perspectiva de gestão de mobilidade (por exemplo, optimização do processo de "handover"), quer de uma perspectiva de gestão de recursos (por exemplo, realizando um equilíbrio de carga mais inteligente), ou a partir de uma perspectiva de encaminhamento (por exemplo, tornando o encaminhamento mais robusto). Antecipar movimento permite seleccionar caminhos que tenham maior probabilidade de fornecer robustez em topologias voláteis.

Esta dissertação contribui para o tema da estimação da aplicabilidade da mobilidade no contexto da gestão da mobilidade, através de: i) análise e proposta de funções de estimação da mobilidade; ii) integração das funções de utilidade desenvolvidas em uma aplicação de software existente (NSense); iii) validação das diferentes funções em bancada de testes.

Palavras-chave: gestão de mobilidade; padrões de movimento social; redes centradas no utilizador.

Contents

List of Tables

List of Figures

Nomenclature

AP Access Point

DMM Distributed Mobility Management Working Group

Handover the process of transferring an ongoing call or data session from one attachment location to another.

HIP Host Initiation Protocol

IETF Internet Engineering Task Force

IoT Internet of Things

MCF Mobile Coordination Function

MIP Mobile IPv4/IPv6

NTP Network Time Protocol

OS Operating System

PoIs Points of Interest

QoE Quality of Experience

QoS Quality of Service

RWP Random Way Point

SIP Session Initiation Protocol

UE User-equipment

VoIP Voice over IP

Wi-Fi Wireless Fidelity

1 Introduction

Movement estimation techniques are today applicable in different wireless and cellular environments, to assist the network operation in aspect such as routing and resource management. In cellular networks, several attempts to estimate movement have been applied. For instance, fast handover anticipation techniques have been a topic extensively addressed within the context of the IETF.

Moreover, the introduction of personal devices with sensorial capabilities, such as smartphones, and several initiatives to collect large amounts of traces [9, 10] lead to the understanding that devices' roaming behavior is related with the social behavior of users [11, 12]. Such analysis of social behavior is the basis to develop proximity-based services and to be able to estimate movement patterns, both from an individual and a collective perspective. Being capable of estimating such behavior is relevant to optimize aspects of the network operation, be it from a mobility management perspective (e.g., handover optimization), from a resource management perspective (e.g., performing a more intelligent load-balancing), or from a routing viewpoint (e.g., improving routing robustness by selecting a priori paths that have a chance to be more stable when nodes move [13, 14]).

This dissertation contributes to the topic of mobility estimation, in the context of mobility management in mobile networks. It addresses challenges concerning the design of simple solutions on the end-user side that can assist the network operation, by estimating potential handover targets.

For that purpose, the dissertation goals are three-fold: i) to conceive and to validate, derived from a prior concept [8], novel attachment point ranking functions; ii) to validate such functions under realistic settings (testbed); iii) to implement the utility functions in the existing open-source middleware NSense [7].

The remainder dissertation is organized as follows. Still in this section we introduce proposed challenges and goals, as well as the proposed work plan. Section 2 covers state of the art concerning aspects such as mobility estimation in cellular and wireless networks. Section 3 computationally describes the heuristics proposed, while Section 4 is dedicated to implementation aspects. Section 5 covers experimentation including methodology and results achieved. The dissertation is concluded in section 6, where guidelines for future work are also provided.

1.1 Research Questions and Goals

The dissertation focuses on the following research questions:

1. How efficient can an estimation mechanism solely based on end-user roaming behavior inference be?

2. Which indicators (derived from wireless overhearing parameters) should be considered in order to improve inference of preferred attachment points?

3. In terms of performance evaluation, what is the gain derived from applying mobility estimation (throughput, reachability time, end-to-end delay)?

The goals originally proposed to work the research questions are:

- **Goal 1:** to propose, based on existing work [8], ranking functions that rely both on passive and active probing.

- **Goal 2:** to implement the proposed heuristics, in order to better understand operational aspects derived from limitations imposed by technology to wireless overhearing, on existing software (MTracker [15]), and to integrate such implementation into the open-source NSense middleware.

- **Goal 3**: to validate the performance of the implemented solutions in a local testbed.

1.2 Expected Results

To work upon the proposed goals, the following aspects have been proposed:

- Analysis of related work, including the MTracker tool and other similar tools.

- Development and validation (testbed) of the tool.

- Proposal of additional parameters/improved utility functions for inference of roaming.

- Demonstration of the achieved prototype.

1.3 Work Plan

The dissertation work plan comprises five main activities, for which a Gantt chart is provided in Figure 1. The activities are:

- **Activity 1: Testbed setup & state-of-the-art analysis - application scenario, assumptions and requirements; analysis of issues.** This activity has as main purpose to analyze existing code; required setup, and related work.

- **Activity 2: Mobility inference improvement (utility functions analysis and improvement).** Existing and novel utility functions for performing visited network selection have been conceived during this activity. A specific set of indicators to be used in the utility functions has also been analyzed.

- **Activity 3: MTracker 3.0 development**. This activity is dedicated to code specification and implementation. The code has been developed following a modular software architecture and has been integrated into the existing NSense middleware.

- **Activity 4: Performance validation**. This activity concerns validation of the different functions for selected scenarios, as described in section 5.

- **Activity 5: Dissertation.** This activity concerns the dissertation wrap-up and writing, including presentations derived from the dissertation process.

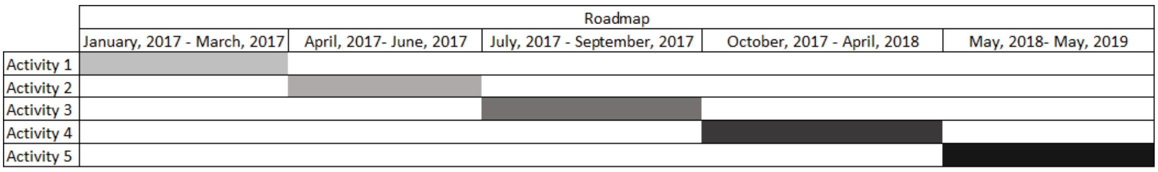


Figure 1: Proposed roadmap.

2 State of the Art

Nowadays, due to an increase of wireless networks and devices connected to it, there is the need to ensure better control and resource management mechanisms. A fundamental part of this whole process is carried out by mobility management. *Mobility management* is a network service which has the purpose of supporting end-user services, e.g., Voice over IP, while users are on the move. This section is therefore dedicated to state of the art on mobility management. It starts by explaining the basics of mobility management, and then goes to mobility modeling and estimation.

2.1 Mobility Management

For years, researchers and scientists have been given the task of defining functions that must be carried out in the context of mobility management, since it is a fundamental part to ensure adequate connectivity and operation of a network. Mobility management as a service has first been introduced in the context of cellular networks, with the aim to assist session handover between different attachment locations [16]. Mobility management mechanisms have been initially worked from a centralized perspective, as the control of mobility was on the network side: in a centralized mobility management architecture, a *mobility management entity* is responsible for managing previous and current status of mobile nodes that are associated with it. Therefore, several protocols have been proposed to support mobility management in the perspective of different OSI stack layers: Session Initiation Protocol (SIP), Mobile IPv4/IPv6 (MIP), Host Initiation Protocol (HIP).

With the Internet evolution towards decentralized service support, mobility of heterogeneous devices increased, and therefore, new paradigms had to be introduced to support mobility from a large-scale perspective. For instance, in *user-centric networking* [17], end-user devices such as smartphones are also networking nodes. Hence, mobility management requires a distributed support to be able to guarantee a better network performance. Decentralization of mobility management is being extensively worked upon by the IETF Distributed Mobility Management Working Group (DMM) [3, 1].

2.1.1 Mobility Management Basic Definitions

In accordance with different functionalities contemplated in mobility management, it is relevant to classify different aspects of mobility management. On the one hand, there is *seamless mobility*, where devices change from a network attachment point to another without interrupting the session or service being provided. On the other hand, there is *nomadic mobility,* where service is stopped until the user connects to a new attachment point [18]. *Roaming* is the ability of a user to access an attachment point (e.g., a wireless AP) outside their network based on their profile.

Throughout this work, roaming denotes the ability of a mobile user to connect to different wireless APs. Chen et al. provide an extensive classification and categorization of different mobility management aspects [3].

Albeit different existing protocols attempt to support mobility management on a specific TCP/IP Layer, mobility management is in fact a cross-layer process, as shown in Table 1.

Table 1: Data plan, basic functions [3].

TCP/IP stack model layers	**Basic functions in mobility management**
MAC layer	• Provides mobility management related with physical signal detection and measurement, which can be used for function and performance optimization.
Network layer	• Provides terminal mobility within a specific network segment. • Provides necessary information about link status and L2 (Layer 2) handover starting/finishing event notification, which can be used for function and performance optimization.
Transport layer	• Provides mobility independent of the lower-layer protocols and physical transmission media, and transparent to the upper layers. • Mainly supports terminal mobility and network mobility. • Provides L3 (Layer 3) handover starting/finishing event notification to the upper layers for handover performance optimization.
Physical layer	• Provides end-to-end mobility support for sessions.
Application layer	• Provides various types of mobility support, especially for high-level mobility (personal mobility and service mobility).

2.1.2 The Handover Process

Short-range wireless technology such as Wi-Fi and Bluetooth, are today a commodity in any personal equipment. Moreover, Wi-Fi is in fact, the technology that complements any type of Internet access. Personal devices such as smartphones, as well as a variety of IoT devices receive regularly a high number of *beacons* from different Wi-Fi APs, i.e., they *overhear* wireless data, even if they do not perform an attachment to specific APs. This brings in the possibility to exploit parameters that wireless devices broadcast. Such information can assist multiple aspects, e.g., better understanding crowd roaming aspects [19]; detecting PoIs in a non-intrusive way, simply based on wireless beacons that devices get anyway [20].

On the other hand, it is also possible to rely on active probing the APs that serve different networks in the range of mobile devices, with the intention of determining the quality of potential connections to APs and as a result, decide which AP to connect to [21]. Collecting measures of bandwidth, amount of traffic that is blocked or redirected, as well as the time to receive a reply from external servers are some of the parameters used to determine the QoS provided by a specific AP, and these parameters can be used to rank such AP, from the perspective of a specific UE.

The IEEE 802.11 standards establish that the association to a “best” AP is, by default, computed based on signal strength (from the AP), but some authors consider that this parameter may not be enough to define a usage preference, as it does not reflect the QoE that the user experiments. Hence, related work proposed to introduce more information on the status of an AP in the beacon frames and probe response frames [22]. The authors demonstrate that the extra data was not significantly higher, and a better use of the resources was found in the network.

2.1.3 Mobility Management Main Functions

The mobility management process itself covers two main functions:

- Handover (also coined handoff) management.
- Location management.

Figure 2 illustrates the different aspects worked upon both for handover and location management.

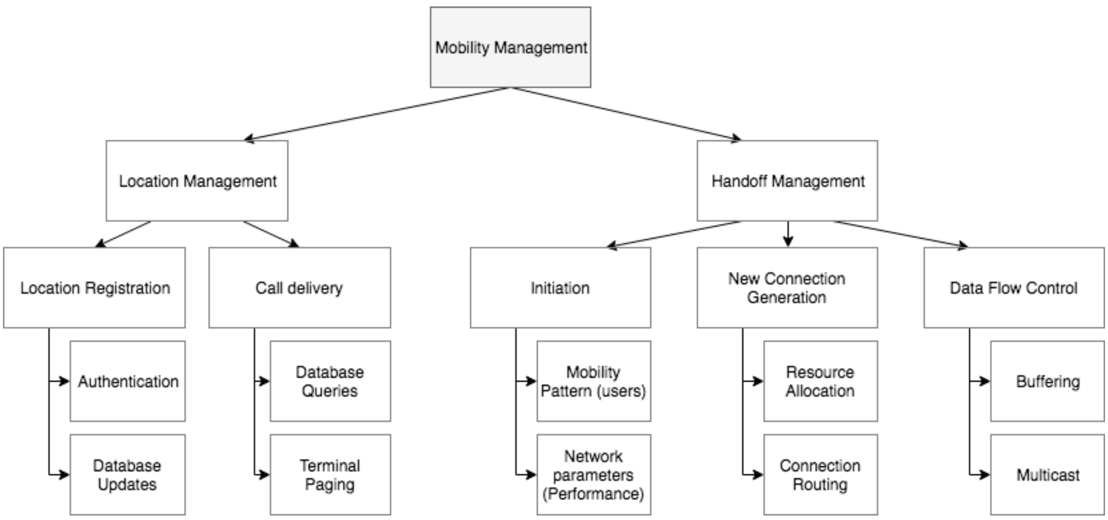


Figure 2: Mobility Management [4].

2.2 Mobility Modeling and Estimation

2.2.1 Mobility Models

Mobility models assist in the definition of both individual and collective movement. There are several mobility models, ranging from *synthetic* models, i.e., derived from obtained traces, as well as models that attempt to describe potential types of movements [23, 24, 25, 26]. The most common ones are: RWP, Manhattan, as well as *social mobility* models [27]. Mobility modeling is relevant not just to emulate a network behavior. Specific models can assist operations performed on the network if applied together with estimation mechanisms, capturing information about the environment or predicting the movement of connected nodes. In this case, networking tasks such as handover, resource control or routing can benefit from the application of mobility modeling. For instance, the handover process can be optimized by inferring behavior about how and when mobile nodes move from one attachment point to another. In the same way, resource management can be improved in advance, e.g., by providing better support to the next attachment point of a node. Similarly, in wireless networks it is possible to assist routing by studying how people (and carried wireless devices) interact with each other.

History of behavior is therefore a first aspect to build a mobility estimation mechanism from. In more recent work, Wang et al. propose a framework with neural networks [28] to study the UE's roaming behavior within defined clusters, and thus recognizing the best attachment points for that cluster. While relevant, this first approach to integrate machine learning into mobility management has been developed with simulations, and therefore, did not address implications derived from implementation aspects.

2.2.2 Tracking Mobility Patterns

Another relevant line of work concerns estimating "best" attachment points by capturing prior individual and collective roaming behavior and attempting to understand where the nodes roam to [29, 30] in a way that is not intrusive. This is what the MTracker [8][1] middleware does, for instance. The MTracker passively relies on overheard wireless information derived from visits to wireless APs while a user roam. This is information that today is available in our smartphones on the list of visited networks. The MTracker computes specific heuristics for indicators such as duration of visits, and "preferred" (more visited) networks and estimates a ranking preference based on the user's roaming habits. This ranking and potential time-to-handover is then passed to a function on the network, which makes a decision on whether or not to perform a handover [29].

The MTracker solution is relevant to our work and is better described in section 4.

2.3 Intermediate Findings' Discussion

As discussed in the prior sections, mobility management is a continuous and relevant field of study regarding network architectures of the future. Most solutions provided have been devised following a centralized architecture and did not take into consideration the possibility to integrate estimation aspects to improve both handover and location management.

Still, on the functioning of both wireless and cellular networks, it is feasible to develop heuristics that rely on small data overheard by devices, and to estimate better where nodes

[1] https://github.com/COPELABS-SITI/ULOOP-MTracker

are going to move to. This is an aspect that this dissertation addresses, specifically trying to understand which functions could be applied and the impact in performance.

3 Computational Aspects, Ranking Functions

This section covers the utility functions developed during this dissertation to perform ranking of preferred networks, and provides the list of selected indicators, as well as the derived utility functions.

3.1 Ranking Indicators

Tables 2 and 3 describe the different indicators considered in the definition of ranking functions. The selection of indicators took into consideration both *passive* and *active* measurement aspects.

In passive measurement, we took into consideration indicators which can be obtained via overhearing, in a non-intrusive way. In active measurement, we took into consideration indicators collected via network from probing.

Indicators obtained via passive measurement bring in advantages in terms of requiring less processing time to be acquired. While indicators derived from active measurement may result in more accurate measurement at the expense of a larger overhead.

3.1.1 Passive Indicators

Table 2: Indicators selected to perform ranking via passive probing.

Indicator	**Name**	**Definition**	**How is it computed**	**Coding aspects**
$v_{ij} \in [0,\infty]$	Visit	A visit from node i to node j implies that node i is authorised (by j) to use its networking resources.	Each time a station performs a successful IP attachment to an Access Point.	WifiManager indicates when a new connection is established.
$V = \sum_{j=0}^{n} v_{ij}\ j \in [0,n]$	Total visits	Number of visits that node i does to node j over time.	Each time a station performs a successful IP attachment to an Access Point, the counter v is incremented.	Each time that WifiManger reports a new connection, sucht event is saved in data base.
$d_{ij} \in [0,\infty]$	Visit duration	Time interval (seconds) since node *i* is authorised by node j to be attached, until node i detaches.	d is computed via the difference of timestamp in and timestamp out.	When a new connection is performed, the time is saved in the data base as well as when the connection is finished.
$d_{avg} = \alpha * d_{avg} + (1-\alpha) * d_{ij}$	Average duration of a visit	Time interval (seconds) that node i is in average attached to node j, based on an exponential moving average formula.	Each time a visit starts, davg is computed based on the proposed formula. α has to be adjusted, derived from specific measurement.	This value is calculated each time there is a new connection and it is saved in data base.

Table 2: Indicators selected to perform ranking via passive probing.

Indicator	**Name**	**Definition**	**How is it computed**	**Coding aspects**
$a_{ij} \in [0,1]$	Visited network attractiveness	A parameter that a user sets by hand (e.g. gives more preference to using network1 than network2) or it can be passively collected via, e.g., distributed trust schemes that are present in the network (e.g. provided by the operator).	a) The user is provided with a deterministic scale (1,2,3,4,5), selecting manually the value of a. b) The user is provided with a continuous scale (0-100), selecting a specific value based on own preference for the network.	This is a numeric value provide by the users through user interface and it is saved in data base.
$rej_{ij} \in [0,\propto]$	Rejected visits	Number of times a node i is not authorised by node j to access its resources.	Each time a IP authorization request is rejected, the device increments the counter.	WifiManager shows when a MAC is requested and when it is rejected.
$te_{ij} \in [0,\propto]$	Visit gap	Time gap (in seconds) since the last visit from node i to a specific visited network j.	Based on timestamp difference.	Those values are saved in the data base when a new connection is stablished and finished.
$n_i \in [0,\propto]$	Node degree	Number of peers around a device i, at instant t.	Based on regular Wi-Fi scanning, we obtain the number of peers at instant t.	WifiDirect API provides functionalities that shown the devices nearby.

The following passive indicators have been considered:

- Visit v_{ij}. Each time a device i connects to an access point j corresponds to a visit.
- Number of visits v. V corresponds to the sum of all visits since the application started.
- Duration of a visit (d_{ij}), corresponds to the total duration in seconds for a visit.
- Average visit duration, based on an Exponential Moving Average (EMA) of d.
- Attractiveness of an AP, a_{ij}, a parameter manually set by the user, to assist in reaching faster a level of preference for visited networks.
- Number of rejected visits rej_{ij}.
- Gap in time between visits to an AP j, te_{ij}.
- Node degree n_i.

3.1.2 Active Indicators

Table 3: Indicators selected to perform ranking via active probing.

Indicator	Name	Definition	How is it computed	Coding aspects
$a_{ij} \in [0,1]$	Visited network attractiveness	A parameter that a user sets by hand (e.g. gives more preference to using network1 than network2) or it can be passively collected via, e.g., distributed trust schemes that are present in the network (e.g. provided by the operator).	a) The user is provided with a deterministic scale (1,2,3,4,5), selecting manually the value of a. b) The user is provided with a continuous scale (0-100), selecting a specific value based on own preference for the network.	This is a numeric value provide by the users through user interface and it is saved in data base.
$rej_{ij} \in [0,\infty]$	Rejected visits	Number of times a node i is not authorized by node j to access its resources.	Each time a IP authorization request is rejected, the device increments the counter.	WifiManager shows when a MAC is requested and when it is rejected.
$n_i \in [0,\infty]$	Node degree	Number of peers around a device i, at instant t.	Based on regular Wi-Fi scanning, we obtain the number of peers at instant t.	WifiDirect API provides functionalities that shown the devices nearby.
$c_{ij} \in [0,1]$	Internet access availability	Boolean value. 1 if a ping to a server result; 0 otherwise.	A HTTP request is executed to verify if there is internet connection.	HttpURLConnection is used in order to create the connection to the external server.
$q_{ij} \in [0,4]$	Signal strength level	Deterministic approach to model signal strength. a single integer from 0 to 4 representing the general signal quality. This may consider many different radio technology inputs. 0 represents very poor signal strength while 4 represents a very strong signal strength.	Each time a Wi-Fi scan is performed, qij is computed.	WifiManger is used to obtain the values from access points every time that a new scan is performed.
$dr_{ij} = \frac{f}{T}$	Data rate	The data rate (bps) based on the transmission of a file with f MBytes, over the total download time T for that file.	When a connection is established a file [2] is downloaded from a Web service.	java.net. URL allows to connect to an external service and download the file. Since the process start, time is counting until the task is finished.

2 http://www.ovh.net/files/10Mb.dat.

Table 3: Indicators selected to perform ranking via active probing.

Indicator	Name	Definition	How is it computed	Coding aspects
$p_i \in [0,]$	IP-based peers	Number of stations connected to AP j (with an attributed IP) at instant t.	A ping is executed to each IP address on the network. This is executed just in the actual sub-network.	Using Java.net functionalities is executed a ping to the IP address.
$r_{kj} \in [0,1]$	Ranking of the preferred AP for peer j	Neighbor recommendations for preferred AP. <Neighbor sends AP identifier (hashed MAC) and respective ranking.	Neighbor broadcasts its preferred AP, via Wi-Fi Direct.	DNS service provided by Android, is used in order to exchange information via TXT Records.

In what concerns indicators derived from active measurement, our selected set is:

- Internet Access availability c_{ij}. A HTTP request is executed to verify if there is internet connection.
- Signal Strength level q_{ij}. Each time a Wi-Fi scan is performed, q_{ij} is computed, and its value represent the strength of the AP.
- Data rate (dr_{ij}). When a connection is established a file is downloaded from the web service http://www.ovh.net/files/10Mb.dat
- IP based peers (p_i). A ping is executed to each IP address on the network. This is executed just in the actual subnetwork.
- Attractiveness of an AP, a_{ij}, a parameter manually set by the user, to assist in reaching faster a level of preference for visited networks.
- Ranking of the preferred AP for peer j. Neighbor broadcasts its preferred AP, via Wi-Fi Direct.

3.2 Ranking Functions

As explained, our work follows the work developed in the context of the MTracker middleware. The MTracker has been designed to integrate any utility function to rank visited networks. The ranking functions developed in this dissertation follow the same methodology. We consider an equation r_{ij} as corresponding to the ranking (cost) that node i computes towards the network controlled by node j. For each function, we have considered both the immediate computed r_{ij} as well as the computation with history, based on an exponential moving average of the cost r_{ij} as provided in Equation 1. By relying on a exponential moving average function where

$r_{ij_{t-1}}$corresponds to the last computed value for r_{ij} and r'_{ij} stands for the instant computation of r_{ij}. By tuning α one shall be providing more weight to more recent or to older instances of r_{ij}.

$$r_{ij} = \alpha * r_{ij_{t-1}} + (1-\alpha) * r'_{ij},\ \alpha \in [0,1] \tag{1}$$

In the next sections the functions are discussed.

3.2.1 MTracker Benchmark Function

The original MTracker considered a single equation r_{ij} based on passive measurement. The rationale of this function, provided in Equation 2 is: the longer and the more often a node visits a specific network, the higher the preference of that network to the node, provided that such visits are recent. Such function has been designed to have enough sensitivity to distinguish between targets that seem to be preferential (for instance, high a_{ij} and long d_{avg}) but that have actually been heavily visited a long time ago (long te_{ij}). The function also takes into consideration the number of rejected connections rej_{ij} against the total number of visits v.

$$r_{1_{ij}} = a_{ij}^2 * (\frac{\sqrt{d_{avg}}}{te_{ij}+1})^{\frac{v}{rej_{ij}}}\ a_{ij} \in [0,1] \tag{2}$$

3.2.2 Ranking Utility Functions based on Passive Measurement

A first set of functions based on passive measurement indicators has been derived from the original MTracker function (r_1) provided in Equation 2. r_2, provided in Equation 3, relies on the rationale that the longer the duration of visits and the smaller the interval between visits (te_{avg}), the better the ranking. The function is quite similar to Equation 2, being the main difference the fact that this function counts with the time gap between visits, and its weight in comparison to the average duration of visits. For instance, if in average visits are long for node A and short for node B, but if the interval between visits for node A is also much larger than for node B, r2 shall consider such variation, while Equation 2 will not.

$$r_{2_{ij}} = a_{ij}^2 * \sqrt{d_{avg}} * (\frac{d_{avg}}{te_{avg}+d_{avg}})^{\frac{rej_{ij}}{v}},\ a_{ij} \in [0,1] \tag{3}$$

A third function is r_3, provided in Equation 4, which considers recommendations from neighbors concerning their preference for node j, an AP, i.e., it considers the **degree centrality** of AP j. The rationale for this function is that the more popular a node j is from the perspective of neighbors of i, the higher the ranking of this AP for node i. Hence, the higher the centrality of j, the higher r_3 will be.

$$r_{3_{ij}} = (1 + \frac{\sum_{k=0}[r_{kj} - r_{ij}]}{1+[(r_{ij}-1)*(r_{ij}-2)]}) * a_{ij}^2 * \sqrt{d_{avg}} * (\frac{d_{avg}}{te_{avg}+d_{avg}})^{\frac{rej_{ij}}{v}},\ a_{ij} \in [0,1] \tag{4}$$

3.2.3 Ranking Utility Functions based on Active Measurement

For the set of utility functions that consider active measurement indicators, we selected 3 possibilities. r_4 provided in Equation 5 considers the quality level of the connection, and is a function of the quality of the connection, both at the MAC Layer (provided by the value q) as well as at the IP layer (provided by t_{ij} and $(p(i))$. The rationale for this function is that the better the quality of the channel and the lesser the number of neighbors around $(p(i))$, the better the ranking r_4 is.

$$r_{4_{ij}} = \frac{c_{ij}}{1+p_i} * dr_{ij} * \frac{q}{4} \tag{5}$$

On a second embodiment, we consider r_5 (cf. Equation 6) where recommendations provided by neighbors are considered to rank node j from the perspective of i. The total number of neighbors that prefer j is computed based on the parameter z_j. The rationale for this function is that the more preferred j is, the higher its ranking for node i, assuming that the quality of the connection exhibits a good level. Hence, in comparison to function r_4, this function brings in the possible weight of recommendations.

$$r_{5_{ij}} = \frac{c_{ij}}{1+n_i} * dr_{ij} * \frac{q}{4} * log(n_j + 2) \tag{6}$$

The final function considered is r_6, where we used recommendations. Instead of choosing the ranking based on the number of recommendations from neighbors for a preferred AP, function r_6 (cf. Equation 7) considers the ranking r_{kj} from each neighbor k (out of n neighbors) towards AP j. The rationale for this function is that the more preferred j is, the higher its ranking for node i, assuming that the quality of the connection exhibits a good level. This function is therefore quite similar to function r_5; the difference is that the preference towards an AP j is counted via the true ranking and not by the number of recommendations towards that specific AP.

$$r_{6_{ij}} = \frac{c_{ij}}{1+n_i} * dr_{ij} * \frac{q}{4} * log(\frac{\sum_{k=0} r_{kj}}{n+1}) \tag{7}$$

3.2.4 Summary of Ranking Utility Functions

For the sake of clarity, Table 4 summarizes the functions that we have set for validation. Three functions are selected to evaluate ranking on passive measurement (r1, r2, r3), while three functions are based on passive measurement. Out of these, two functions used neighbor recommendations (r5, r6).

Table 4: Summary of utility functions.

Function Id	**Measurement type**	**Rationale**
r_1	Passive	The longer and the more often a node visits a specific network, the higher the preference of that network to the node, provided that such visits are recent.
r_2	Passive	The longer the duration of visits and the smaller the time interval between those visits, the better the ranking.
r_3	Passive with recommendations	The more popular a node j is from the perspective of neighbors of i, the higher the probability of being less congested and hence, of being preferred from the perspective of i.
r_4	Active	The better the quality of the channel and the lesser the number of neighbors around $(p(i))$, the better the ranking r_5 is.
r_5	Active with recommendations	The more preferred j is, the higher its ranking for node i, assuming that the quality of the connection exhibits a good level.
r_6	Active with recommendations	The more preferred j is, the higher its ranking for node i, assuming that the quality of the connection exhibits a good level.

4 Implementation Aspects

This section goes over the implementation developed to provide the ranking functions. Our mobility estimation solution is derived from the MTracker prior code[3] and therefore we have applied a development methodology for code reuse, illustrated in Figure 3.

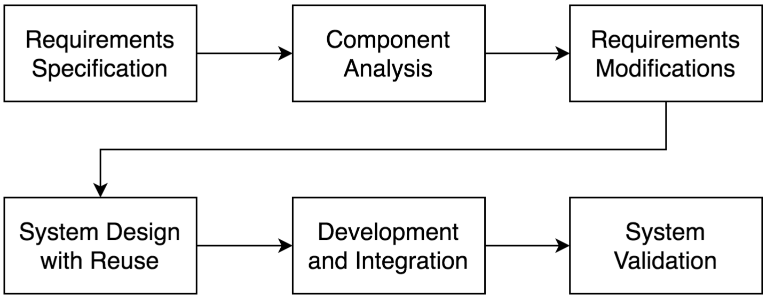


Figure 3: Reuse methodology [5].

As illustrated, we have started with the requirement's specification, including novel functions (section 3) followed by the component analysis (section 4.1). Development and integration were performed afterwards (section 4.2) followed by system validation (section 5).

4.1 Background: MTracker and NSense

4.1.1 MTracker

The *Mobility Tracker (MTracker)* is an open-source end-user mobility estimation tool[4] developed in the context of the IST FP7 ULOOP project [8, 31]. The MTracker has been conceived as a UE plugin which has the purpose to assist centralized mobility management solutions in performing handovers based on the history of use of preferred networks. For that purpose, the MTracker passively tracks anonymous properties of a user's roaming behavior and ranks each visited network based on a specific algorithm which takes into consideration aspects such as number of visits to a given access point and the average duration of such visits. The MTracker application then tries to predict in how much time the node will change the network connection, and which will be the next network.

MTracker has been developed in Android. Within the user side, the MTracker collects information concerning visited networks, periodically computing a ranking to each visited network. Then, periodically, it emits a message to potential anchor points or, in the case of the ULOOP project, to the entity MCF, as illustrated in Figure 4. This is done by having a MTracker server-side plugin on the gateway, aspect which facilitates the future development of MTracker.

[3]https://github.com/COPELABS-SITI/ULOOP-MTracker
[4]https://github.com/COPELABS-SITI/ULOOP-MTracker

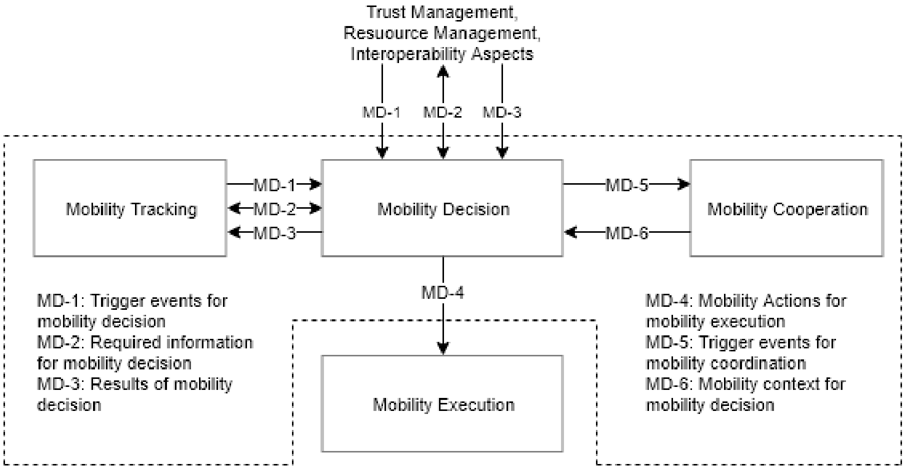


Figure 4: MTracker role in mobility management, project ULOOP use-case [6].

The MTracker input is collected via Wi-Fi overhearing and obtained via background processes every *t* seconds, as illustrated in Figure 5. The aim of the tracking in ULOOP was to provide a simple and yet effective implementation on how to improve mobility management based on estimation, from an operational perspective. The MTracker therefore relies on the regular Wi-Fi scan process and computes the ranking of an AP based on a specific utility function (rf. to section 4.). Such ranking function provides a notion of "best gateway" from an end-user perspective and derived from a specific set of overheard parameters. The MTracker passes the ranking along with an estimate of time to handover to the MCF entity on the network. Therefore, the MTracker leaves the decision of handing over to the MCF entity.

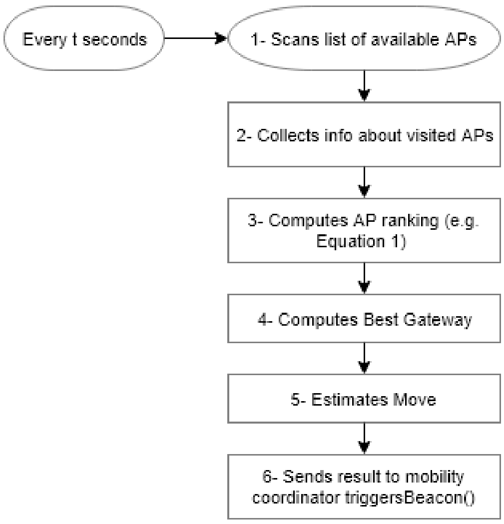


Figure 5: MTracker high-level operation, Mobility management context.

4.1.2 NSense

The *Nearness Sensing (NSense)* open-source middleware [7][5] has been developed to assist in a better understanding of the level of social interaction of users carrying mobile devices. For such purpose, the NSense architecture (illustrated in Figure 6) relies on multiple sensors to gather information which is then classified in order to assist in inferring sociability levels (social interaction). The original NSense classification modules, coined *pipelines*, are location (derived from GPS and from Wi-Fi); proximity (derived from relative distance, Wi-Fi); environmental sound level (derived from the microphone) and motion (derived from the accelerometer data).

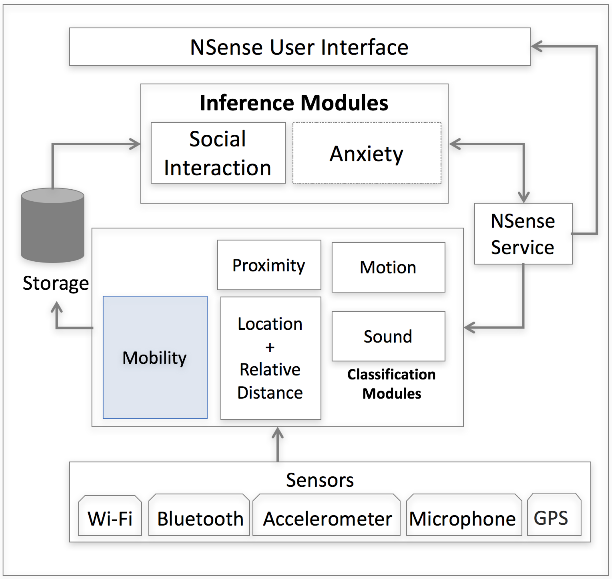


Figure 6: NSense Architecture with the new Mobility pipeline [7].

In this dissertation, our implementation, which is based on the MTracker, was proposed to be integrated into the more recent NSense middleware, under a new pipeline: The **Mobility** pipeline, as described next.

4.2 The NSense Mobility Pipeline

The NSense mobility pipeline is a result of the integration of our interpretation of the MTracker code, into a new pipeline of NSense, the mobility pipeline. The integration and applicability of this pipeline concerns gathering context about the user's roaming habits in a way that is not intrusive and that can assist analysis of social interaction, as well as to boost social interaction derived from learning of roaming habits.

Specifically, users that exhibit similar patterns in terms of roaming habits are good candidates to exchange information and to form communities. By adding such context to NSense,

[5]https://github.com/COPELABS-SITI/NSense

it is envisioned that future versions of this middleware, as well as future studies can benefit from the correlation between mobility data, and social interaction aspects.

Figure 7 shows the integration of the mobility pipeline into the existing code of the MTracker, where blue boxes correspond to new code added to the initial MTracker code.

From an operational point of view, once the application starts (running in background), it checks whether the device is connected to a wireless network, i.e., whether the device has been authorized to use resources on that network. If so, the device starts by validate if the information of the actual AP is saved into the data base. Once validation is executed the information is updated or saved depends if the AP was previously registered or not. Then, every time that Android's Wi-Fi API performs a scan, the parameters of the function selected are calculated, once those values are saved in the data base the next step is validate if the actual AP is the best or not. If so, the application is going to switch to the best AP registered. Once the device is connected to the new AP the entire process is executed again.

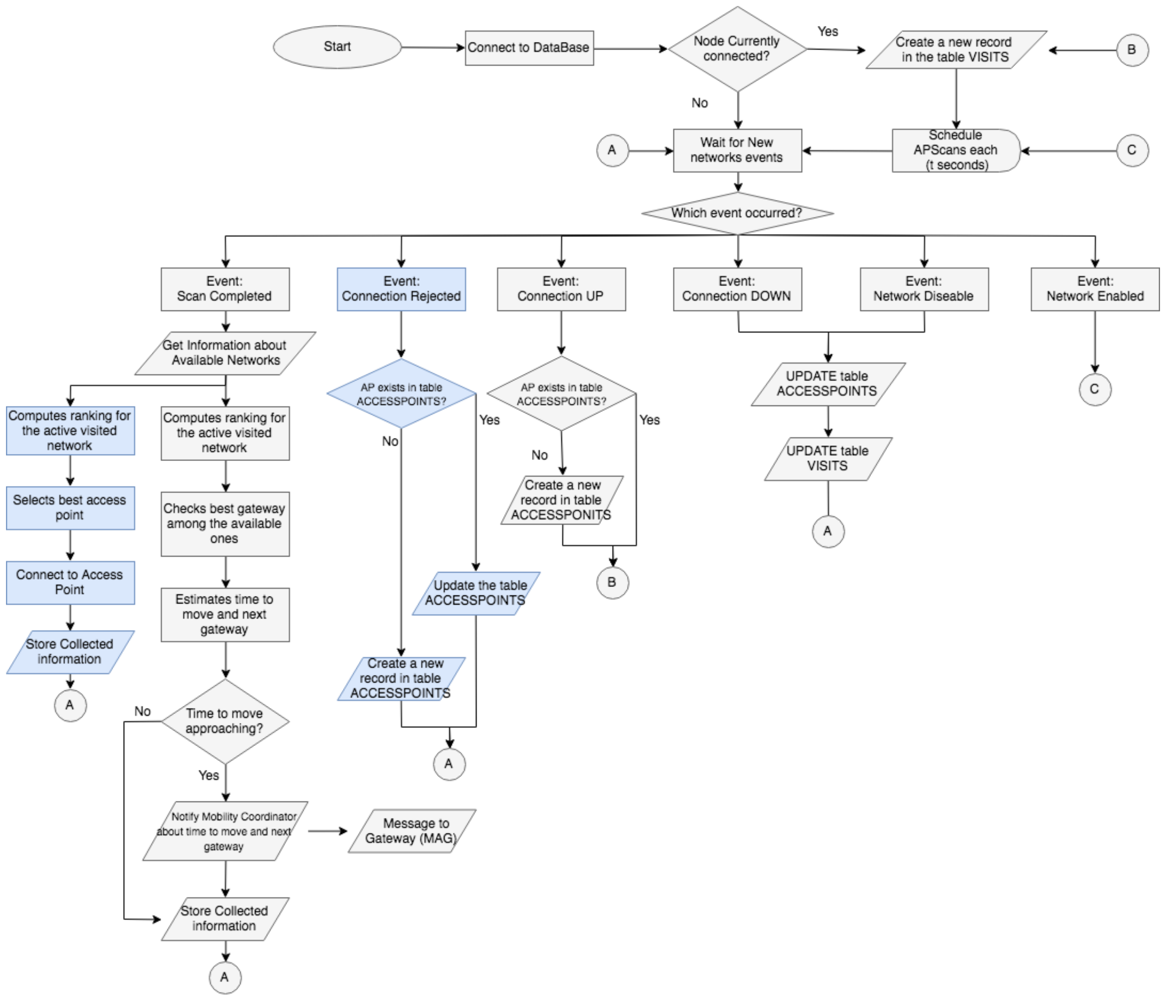


Figure 7: MTracker flowchart [8].

4.2.1 Storage

As mentioned before, the proposed software carries out a series of actions in which it highlights the need to be able to store information product of the behavior to which the software is submitted. That is why in this section we explain how the database is structured and which information is saved in it.

The database contains 2 tables, *Access Points* and *Ra*nking. The *Access Points* table stores information of the APs to which the UE is or has been connected to. The table considers the hashed BSSID of the AP as a key in order to guarantee that only one an entry in the table related to each AP. The table stores, for each AP, information such as number of connections, attractiveness or number of rejections.

The Ranking table stores the ranking values for each AP every time a calculation of this sort is performed. It is important to note that in this table the values are hyperlinked in order to be able to study the behavior of the functions over time. Figure 8 shows how this database is composed.

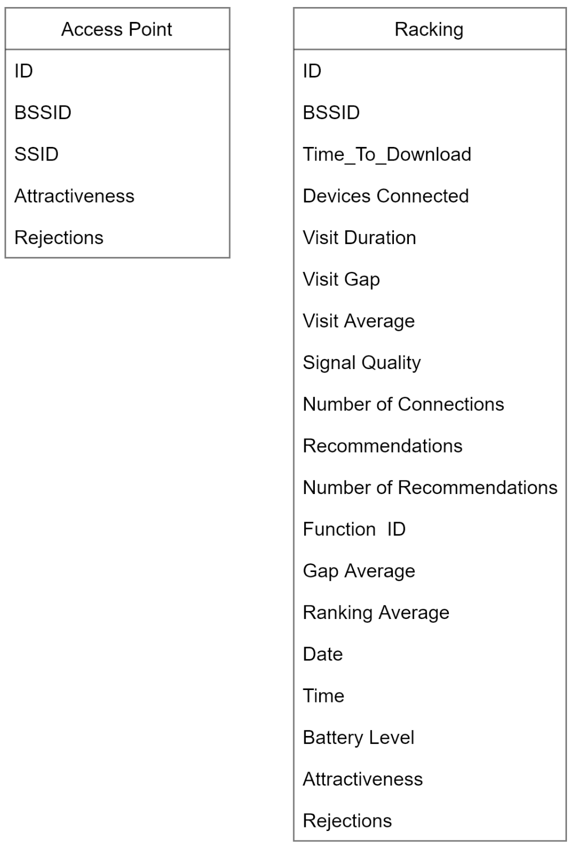


Figure 8: Mobility pipeline database structure.

4.3 Implementation Aspects

The implementation has been developed in Java for Android, as the original MTracker and NSense code is made available for the same platforms. The flow-chart for the implementation is provided in Figure 9.

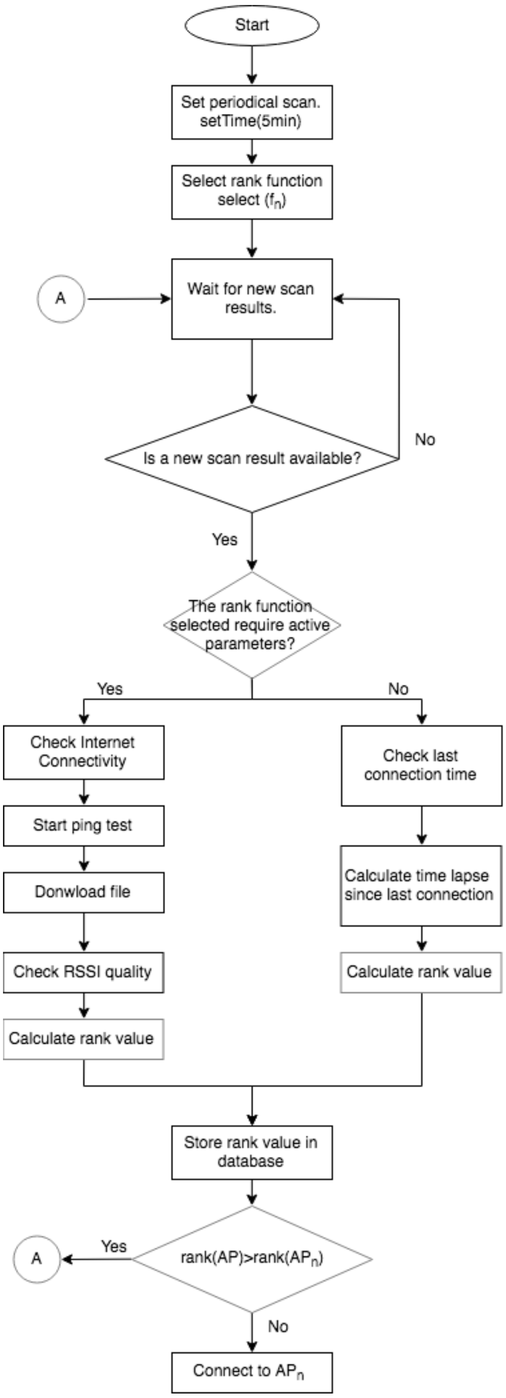


Figure 9: Computational diagram of the Mobility pipeline.

The developed package is selected in the NSense software by the user, and therefore, the Mobility pipeline starts recording the information described, and stores it in the NSense database. To assist the evaluation of the functions, we have added the option to select a specific ranking function, which updates its value for every Wi-Fi scan. While the implementation is currently set this way, it is feasible to change the time window used.

As java does not implemented threading and there was the need to synchronize the devices that were used in the experiments, our implementation recurred to the Android Alarm-Manager library, which wakes up the Mobility pipeline upon function computation. This only guarantees that the commands are executed at the same time in all devices. There is, in addition, a computational delay dependent upon of the OS. Another possibility that could have been considered to synchronize all devices would be to rely on NTP.

All the collected data is kept in an SQLite database.

The code documentation is provided in Annex II.

4.4 Limitations

During the implementation several limitations derived from the use of Android have been detected. A first limitation concerns the fact that when a device cannot attach to an Access Point (not authorized on the MAC Layer), the OS does not provide a message to the respective API. Android provides status information about connections as well as disconnection, but not about a device not being served. This message is relevant to the implementation developed, as the number of rejections that an AP gives to a specific device allows a hint on aspects concerning the channel or connection status.

A second limitation concerns to the use of RSSI for measurements. RSSI is hard coded in Android and therefore, cannot be used directly, as occurs in other OS, to provide a hint on the channel quality. Also, the usage of Wi-Fi API changes between phone makers, what means the library reacts in different ways depending of the device, for that reason in this work were used only Samsung smartphones with the intention of avoid such complexity.

Another limitation is the size of the information that can be transmitted using DNS txt record, this is limited to 88 bytes of data, due to this we only sent relevant information using this mechanism.

4.5 Security and Data Privacy Concerns

Given that, the Mobility pipeline in NSense tracks information via wireless overhearing, there is the need to ensure that data privacy is kept intact, and the user anonymity is kept. Such parameters are, for instance, the MAC address of neighboring devices; duration of visits, etc. The following aspects have been taken into consideration:

- Data collected has been only locally stored and only used for the purpose of ranking APs.

- Information concerning MAC addresses, as well as BSSIDs has been obfuscated via MD5.
- No personal information has been stored locally.

5 Performance Evaluation

This section covers the validation of the ranking functions described in section 3. We start by explaining the performance evaluation settings, having the experiences been performed in the context of a realistic testbed composed of 9 Android devices, as well as of 3 APs. The characteristics of the devices are provided in Table 5. All end-user equipment had installed NSense v4.0[6] (with the new Mobility pipeline).

Table 5: Testbed equipment.

Identifier	Type	Network features	Testbed function
COPELABS_1	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS_2	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS_3	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS_4	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS_5	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS_6	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS_7	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS_8	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS_9	Smartphone Samsung G5. Android Version 6.0	Wi-Fi/Wi-Fi Direct	End-user equipment
COPELABS	Ubiquity access point	802.11a/b/g	Access point public
AP_1	Ubiquity access point	802.11a/b/g	Access point Controlled
freeisg	Ubiquity access point	802.11a/b/g	Access point public
Copelabs_PC	Laptop	Ethernet	Controller

The full topology is illustrated in Figure 10. Where the coverage areas of each access point are overlapped, being the freeisg the one with more coverage area, followed by Copelabs and AP_1 respectively. The smartphones are in place where the coverage areas are overlapped, this is represented by the point “A” shown in the figure.

[6] Available via: https://gitlab.com/citysense_copelabs/NSense/tree/version-3.0

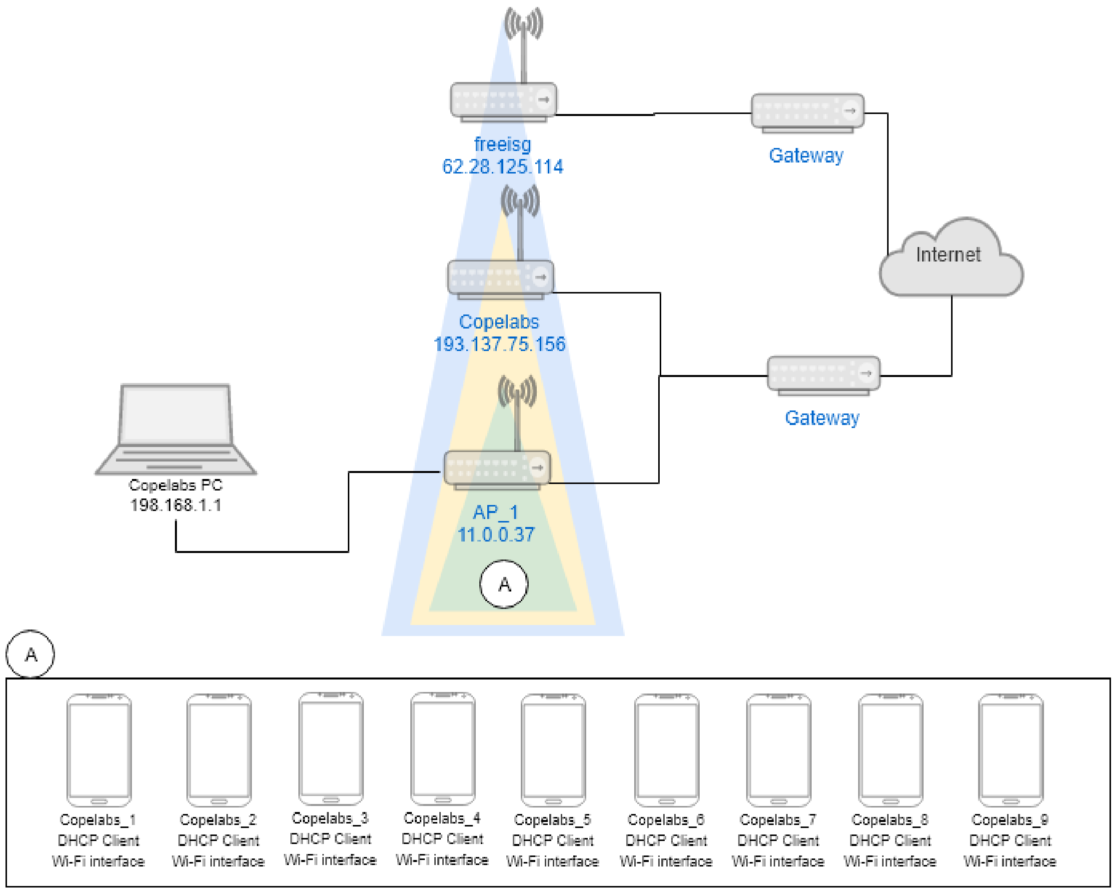


Figure 10: Performance evaluation testbed.

5.1 Evaluation Scenarios

We have considered two different topological scenarios, where conditions have been varied over different days and times. All the experiments have been run from March 2018 until April 2018. The experiments have been run over different days and schedules. We selected two main periods of different days, to start the experiments: 11 a.m. (standing for a "full" period) and 5 p.m. (standing for a "low" usage period).

All experiments have been repeated 5 times, and the raw results obtained are available online [7] and provided in Section 7.

- **Scenario I**. It stands for a small, controlled environment, involving two APs (COPELABS and AP_1). AP_1 (control) is manually started and stopped to create disturbance and to understand the sensitivity of functions. Experiments in this scenario last 45 minutes.

- **Scenario II**. This scenario integrates the full testbed and is intended to emulate a more realistic scenario, given that two APs (COPELABS and freeisg) are connected to the

[7] COPELABS scicommons

Internet, experiencing high load (university campus). Experiments in this scenario last 110 minutes.

Table 6: Scenario II, functions active in end-user devices for scenario II.

Device	Ranking functions considered
Copelabs_1	Function r_2
Copelabs_2	Function r_3
Copelabs_3	Function r_3
Copelabs_4	Function r_4
Copelabs_5	Function r_5
Copelabs_6	Function r_5
Copelabs_7	Function r_6
Copelabs_8	Function r_6
Copelabs_9	No function running

5.2 Evaluation Results

5.2.1 Scenario I, Control Experiments

Results concerning scenario I are provided in Figure 11, where the X-axis represents time, and the Y-axis provides the normalized ranking value. AP_1 was shut down at specific instants in time (1) and then turned on again (2), with the purpose to understand how the different functions can adjust.

In what concerns the functions that perform ranking based on passive measurement (r1, r2, r3), these exhibit a similar behavior. Out of these 3, r1 is the one that exhibits a more conservative behavior, as can be seen at instant 11:15:14, where the function did not adjust well, even though the activity of the AP is quickly recovered. This happens because the rational of this function assigns more influence on the visit gap time than the visit time. That means that as longer the visit time is, as less preferred will be the access point.

In what concerns functions based on active measurement (r4, r5, r6), these functions exhibit a similar behavior in this controlled scenario. They are less reactive to breaks in connectivity, as they use the quality of the connection. Recommendations in this scenario have no significant value, as there were no neighbors around. Hence, the behavior of r5 and r6 had to be similar.

Overall, what can be observed is that functions r2 and r3 (passive measurement) behave well in comparison to functions that require probing, for this very controlled scenario, showing the adaptive capacity to the user behave.

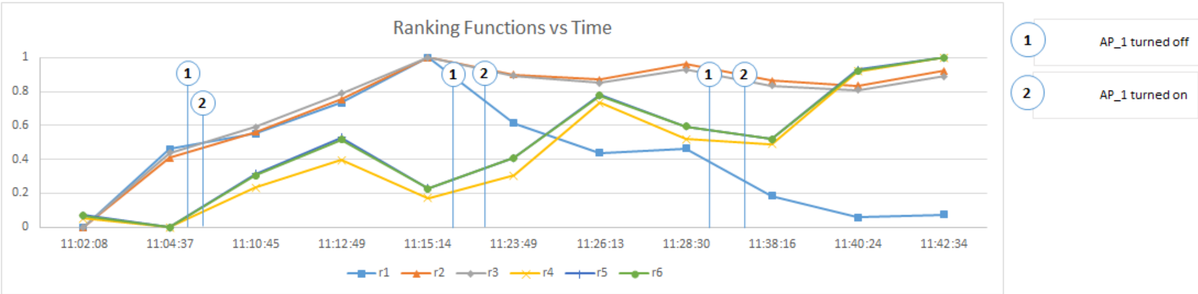


Figure 11: Ranking results over time, scenario I, run I.

The same experiment has been repeated five times in different days of the week, being the respective results provided in Annex I, section 7.1.1. These experiments had as difference the surrounding conditions, as well as a change in the initial value of r.

The experiment has been repeated at different days, afternoon, being the first results shown in Figure 12. Passive ranking based functions r1, r2, r3 exhibit a similar behavior to the one depicted by Figure 11. In what concerns active ranking functions r4, r5, r6, while the behavior is similar in terms of adaptability, there is a difference in the computed values, which we believe is a result of the different connection conditions, for instance, the data rate parameter could be different from one measurement to another.

To understand impact of conditions, this experiment has been repeated five times in different days of the week, being the respective results provided in Annex I, section 7.1.1. These experiments had as difference the surrounding conditions, as well as a change in the initial value of r.

Given that function r1 always exhibited a worse behavior and this function is like r2, we did not consider r1 in the experiments for scenario II.

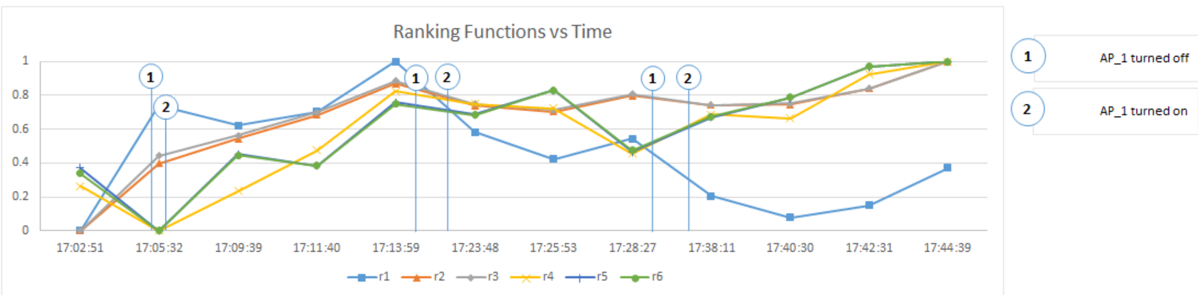


Figure 12: Ranking results over time, scenario I, run I.

5.2.2 Scenario II, Experiments, 11a.m. Period

In scenario II we have considered different experiments per utility function. Results obtained with r_2 are provided in Figure 13. As shown, the device initially has as preferred AP AP_1 (based on the usual sequential behavior of Wi-Fi). At instant 11:15:00 the device, based on user preferences, opts to connect to AP COPELABS. Once the equipment is connected starts to calculate the value of the functions. The result of the function is calculated every 5 minutes. Once the equipment calculates the value of the ranking at instant 11:20:10 decides to switch to AP_1.

After condition (2), where AP_1 does not allow more connections, the function reacts well, and at instant 11:36 COPELABS becomes the preferred AP. As also shown, after instant (4), the device connects to a new existing access point (freeisg), however after 5 minutes of connection, the result of the function selects COPELABS as the preferred one.

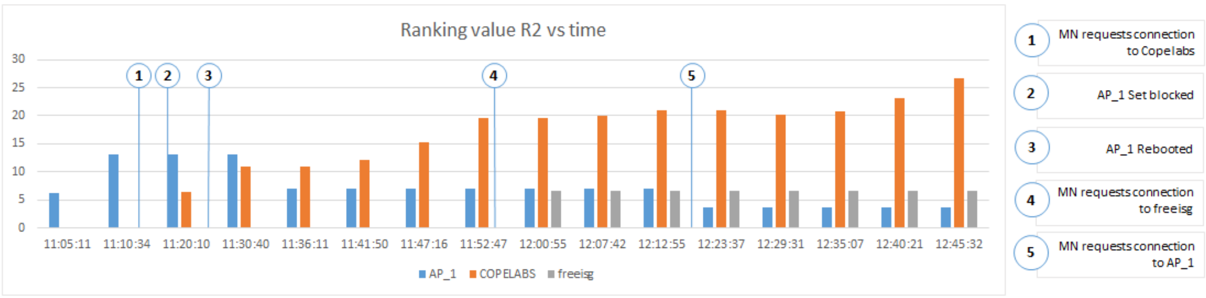


Figure 13: Scenario II, r2 results, perspective of device I.

This experiment has been repeated for each function (cf. Figures 13, 14, 15, 16, 17). Function r2 and r3, which consider the centrality of the preferred AP also from the perspective of neighbors, surprisingly exhibit a similar behavior: while r2 ranking grows with a larger number of devices around the AP as preferred AP, r3 ranking grows with a decrease in the number of devices around the AP as preferred AP. We believe that the similarity in behavior for this case is due to the number of nodes around. Even though both functions have a similar behavior, the values obtained with r3 are smaller than with r2.

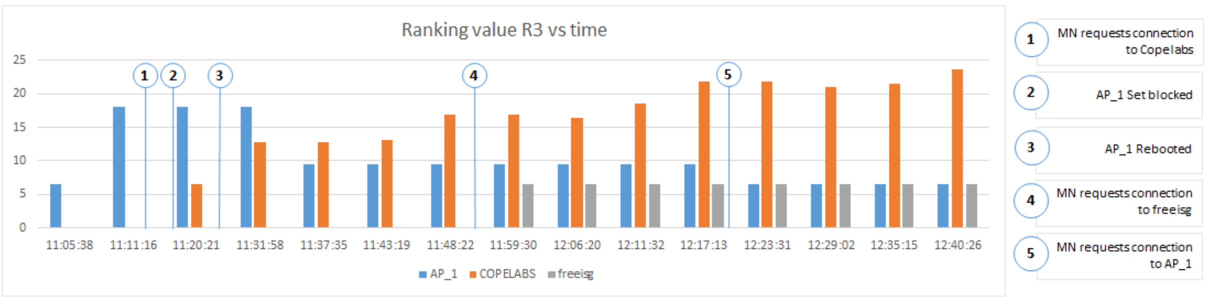


Figure 14: Scenario II, r3 results, perspective of device II.

In what concerns r4, r5, r6 behavior, it can be observed that the more aggressive functions are r5 and r6, which due to recommendations from neighbors creates a more discrepant value for ranked APs. Furthermore, r5 and r6 consider first the connection quality, independently of the prior history. For instance, results for r6 (cf. Figure 17) show that the initial selected AP, AP_1, has been considered in detriment of the available APs around, due to the recommendations provided by neighbors. This seems to imply that while recommendations for a specific AP should be taken into consideration, the significance of the weight of such recommendations need to be better weighted in comparison to prior history of use of preferred APs.

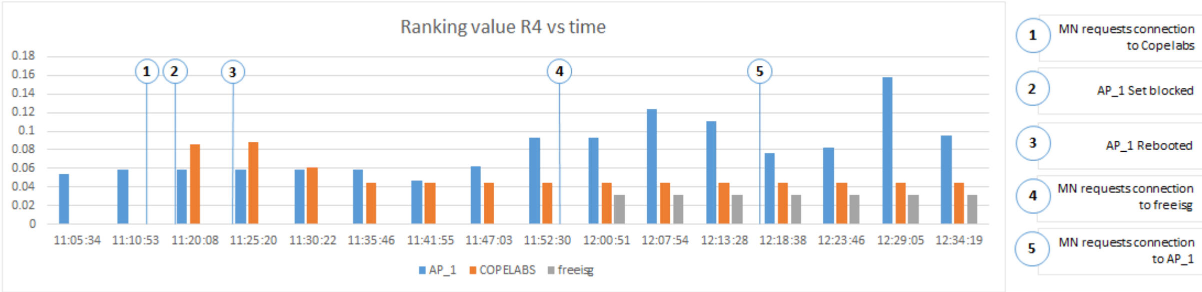


Figure 15: Scenario II, r4 results, perspective of device IV.

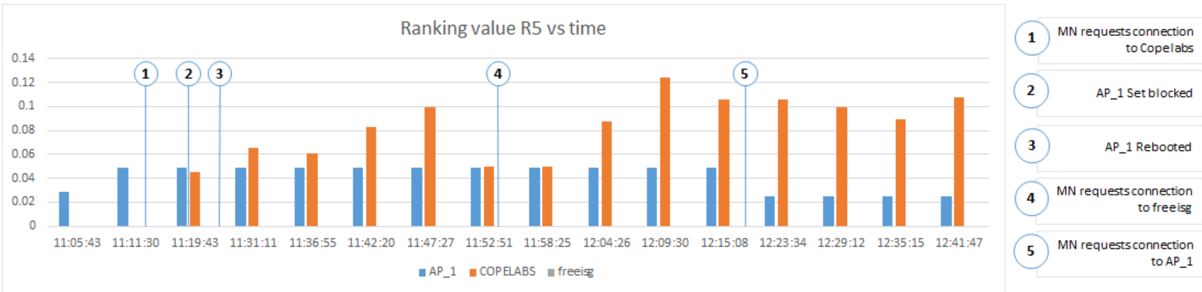


Figure 16: Scenario II, r5 results, perspective of device V.

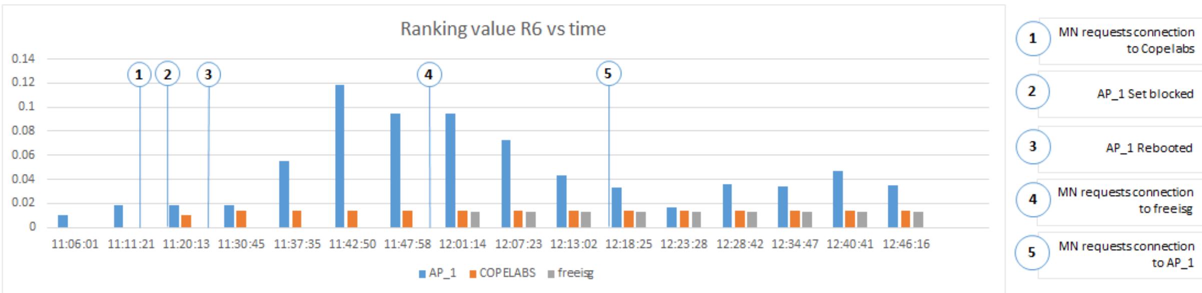


Figure 17: Scenario II, r6 results, perspective of device VII.

5.2.3 Scenario II, 5 p.m. Period

The experiment has been run in the afternoon, to create additional conditions. Results are provided in Figures: 18, 19, 20, 21, 22. While r2 exhibits a similar behavior, r3 exhibits some differences in results, which we believe are due to the differences in neighborhood. However, such differences are not significant. In comparison, r4, r5, and r6 are more sensitive to surrounding conditions. r6 is again the function that exhibits a more variable behavior. This shows that the notion of recommendations, albeit interesting, may generate too much entropy. Out of the three functions, r5 seems to be more stable.

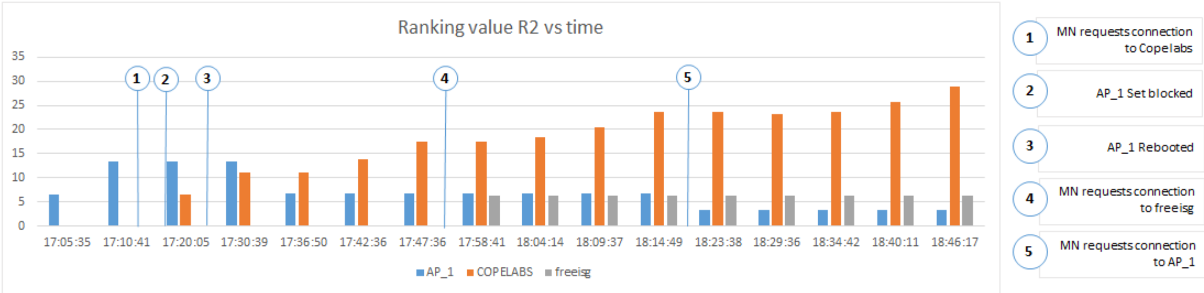


Figure 18: Scenario II, r2 results, perspective of device I.

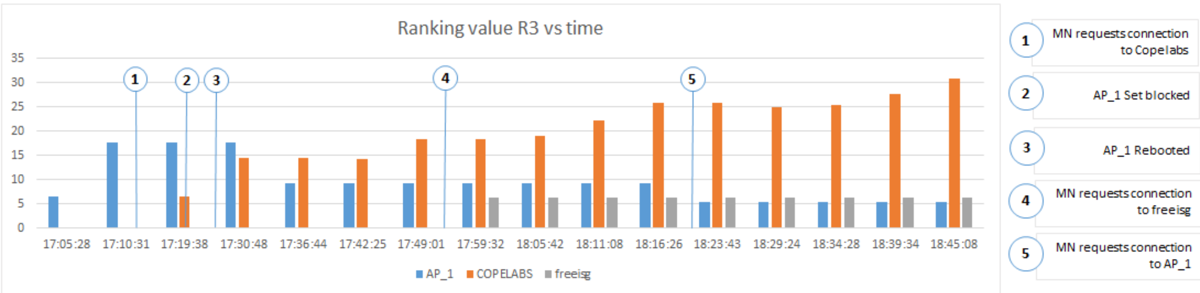


Figure 19: Scenario II, r3 results, perspective of device II.

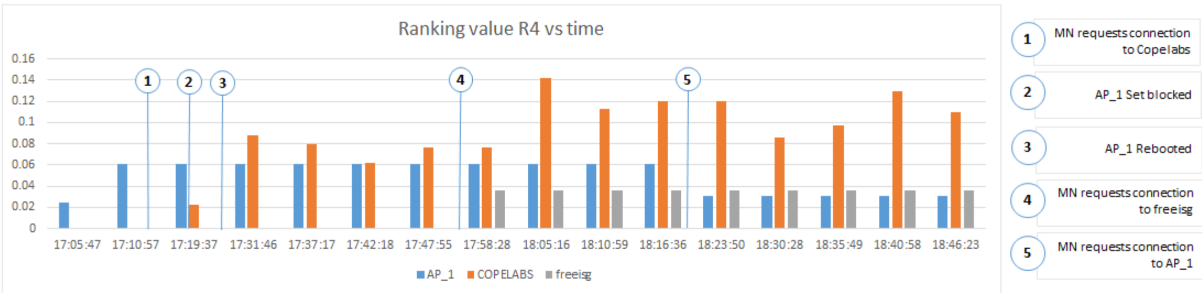


Figure 20: Scenario II, r4 results, perspective of device IV.

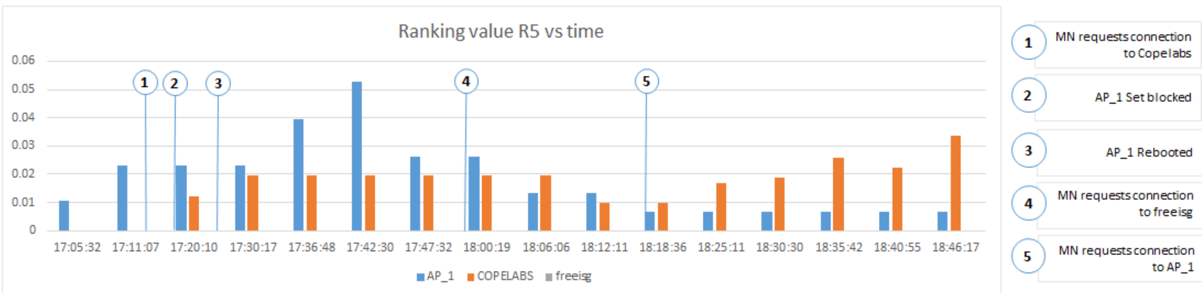


Figure 21: Scenario II, r5 results, perspective of device V.

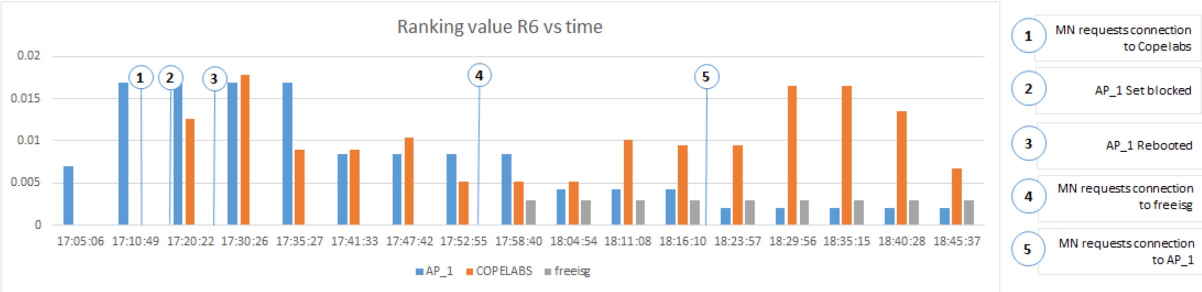


Figure 22: Scenario II, r6 results, perspective of device VII.

5.2.4 Functions' Comparison

In order to provide a performance comparison of the different functions (having discarded r1), we have considered the following values, provided in Table 7:

- **Time to handover**. The period (seconds) it takes for a function to complete handover, from a MAC Layer perspective. Therefore, this period considers: MAC and IP handover.
- **Handover**. Total number of successful handovers performed.
- **Rejected handovers**. Upon request to handover, rejected handovers by the AP.

The functions that complete the handover faster are r2 and r3, i.e., functions that are based on passive measurement. Functions based on active measurement require more complex computation and are as well dependent upon external values (e.g., ping time).

Table 7: Comparison of performance, different functions.

Function	Time Handover(s)	Handovers	Rejected handovers	Total handovers
r2	31	9	10	19
r3	34	9	12	21
r4	59	6	0	6
r5	43	9	11	20
r6	62	10	6	16

5.3 Summary of Results

Based on the performance evaluation provided, this section summarizes the findings, following the initial challenges set on this dissertation:

1. **How efficient can an estimation mechanism solely based on roaming behavior inference be?**

   - We have performed experiments with multiple functions. r1, r2, r3 are based on passive measurement. r2 and r3 are equally relevant and exhibit a good behavior in comparison to functions that require active probing. It therefore has been shown that an estimation mechanism based on roaming behavior inference can be accurate, and relevant to assist different aspects of the network operation. Mobility management solutions can greatly benefit from the integration of functions such as the ones proposed, with little impact in terms of overhead.

2. **What are the parameters that are relevant to consider in order to improve inference of preferred attachment points?**

   - The duration of visits as well as the time gap between such visits is highly relevant to be considered in ranking functions. The rejected number of visits in comparison to the total number of visits is also relevant. Recommendations from neighbors, be it by providing the exact ranking or simply by following a "majority vote" approach, is also relevant to be taken into consideration.

3. **In terms of performance evaluation, what is the gain derived from applying such a mechanism (throughput, reachability times, delay)?**

   - The main gain concerns time to complete handovers, which has impact in terms of both node reachability time and end-to-end delay. Mechanisms such as the ones provided seem to be relevant in terms of fairness. Throughput gains are expected, in cases such as ping-ponging.

6 Conclusions and Future Work

This dissertation explored the application of utility functions to rank preferred networks. The dissertation considered prior work developed, and contributed to such work by:

- Analyzing networking parameters that can be used to develop ranking functions, be it passively (via overhearing) or actively (via probing).
- Suggesting novel ranking utility functions that combine the different parameters.
- Implementing the code to perform such ranking and to provide a history of roaming habits, based on existing code (MTracker), but leveraging it to be integrated into a more recent middleware framework (NSense).
- Developing the testbed to perform evaluation of the proposed functions.
- Performing an evaluation based on the proposed testbed.

The work developed corroborates that mobility estimation based on overheard information can assist significantly the network operation, by improving handover completion time as well as by preventing handover rejections (in case of devices that cannot complete the handover, due to conditions around).

As follow up work, we believe that the proposed functions could be testbed with different mobility management solutions, such as the different MIPv6 solutions, as well as used to assist in contextualization of variable topological environments such as what occurs in mobile crowd sensing environments. For this purpose, our code is publicly available as an NSense pipeline. As it has been developed in an independent way, such code can be easily used in other solutions.

References

[1] D. Liu, J. Zuniga, P. Seite, H. Chan, and C. Bernardos, "Distributed Mobility Management: Current Practices and Gap Analysis," *IETF DMM Working Group, RFC 7429*, 2015.

[2] A. Nascimento and R. C. Sofia, "UMM D3: Report on DMM Related Standardization Activies," Tech. Rep. SITI-TR-12-04, SITI, Universidade Lus{ó}fona and IT/Universidade de Aveiro, 2012.

[3] S. Chen, Y. Shi, H. Bo, and A. Ming, *Mobility Management: Principle, Technology and Applications*. Berlin: Springer-Verlag Berlin Heidelberg, 1 ed., 2016.

[4] I. F. Akyildiz, J. Mcnair, and J. Ho, "Mobility Management in Next Generation Wireless Systems," in *IEEE Wireless Communications*, vol. 87, 1999.

[5] I. Sommerville, *Software Engineering*. Boston, Massachusetts: Pearson Education, Inc, 9th ed., 2011.

[6] Q. Zhou, C. Pampu, F. Sivrikaya, S. Peters, R. Sofia, and O. Marcé, "D3.6:ULOOP Mobility Aspects Specification and Refined Software," tech. rep., 2012.

[7] R. Sofia, S. Firdose, L. A. Lopes, W. Moreira, and P. Mendes, "NSense : A People-centric , non-intrusive Opportunistic Sensing Tool for Contextualizing Nearness," in *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pp. 1–6, IEEE, 2016.

[8] R. Sofia, "A Tool to Estimate Roaming Behavior in Wireless Architectures," *inProc. WWIC 2015*, 2015.

[9] D. Kotz, T. Henderson, and I. Abyzov, "CRAWDAD Trace Repository," *Downloaded from http://crawdad.cs.dartmouth.edu*, 2005.

[10] R. Sofia, L. Inocencio Carvalho, S. D. D. Silva, M. Tavares, and O. Aponte, "Proxemics Data Lab project @COPELABS," 2016.

[11] A.-L. Barabási, "The Origin of Bursts and Heavy Tails in Human Dynamics," *Nature*, vol. 435, pp. 207–211, 2005.

[12] M. Gonzalez, C. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *Nature*, vol. 453, pp. 779–782, jun 2008.

[13] N. Chama, R. C. Sofia, and S. Sargento, "Impact of Mobility on User-Centric Routing," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on.*, jun 2011.

[14] N. Chama, R. C. Sofia, and S. Sargento, “A Discussion on Mobility Awareness of Multi Hop Routing in User-Centric Environments,” Tech. Rep. COPE-SITI-TR-05-15, COPELABS, University Lus{ó}fona, University of Aveiro, 2015.

[15] R. C. Sofia and C. Silva Pereira, “Mtracker v2.0.” SITI-SW-12-07, oct 2012.

[16] S. Chen, Y. Shi, B. Hu, and M. Ai, “Mobility-driven Networks (MDN): From Evolution to Visions of Mobility Management,” *IEEE Network*, vol. 28, no. 4, pp. 66–73, 2014.

[17] R. Sofia and P. Mendes, “User-Provided Networks : Consumer as Provider,” *IEEE Communications Magazine*, vol. 46.12, no. January 2014, 2008.

[18] A. Nascimento, R. C. Sofia, T. Condeixa, and S. Sargento, “A Characterization of Mobility Management in User-centric Networks,” *Smart Spaces and Next Generation Wired/Wireless Networking.*, pp. 314–325, aug 2011.

[19] A. Basalamah, “Crowd Mobility Analysis using WiFi Sniffers,” *(IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 7, no. 12, pp. 374–378, 2016.

[20] R. C. Sofia, L. I. Carvalho, and F. d. M. Pereira, “The Role of Smart Data in Inference of Human Behavior and Interaction,” in *"Smart Data: State-of-the-Art and Perspectives in Computing and Applications". Group, USA. April 2019.* (K.-C. Li, Q. Z. L. T. Yang, and B. D. Martino., eds.), CRC Press, Taylor & Francis, 2018.

[21] A. J. Nicholson, Y. Chawathe, M. Y. Chen, B. D. Noble, and D. Wetherall, “Improved Access Point Selection,” in *MobiSys ’06: Proceedings of the 4th international conference on Mobile systems, applications and services*, (New York, NY, USA), pp. 233–245, ACM, 2006.

[22] M. Abusubaih, J. Gross, S. Wiethoelter, and A. Wolisz, “On access point selection in ieee 802.11 wireless local area networks,” in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pp. 879–886, IEEE, 2006.

[23] D. Karamshuk, C. Boldrini, M. Conti, and A. Passarella, “Human Mobility Models for Opportunistic Networks,” *Communications Magazine, IEEE*, vol. 49, pp. 157–165, dec 2011.

[24] C. Boldrini, M. Conti, and A. Passarella, “Users Mobility Models for Opportunistic Networks: the Role of Physical Locations,” in *IEEE Wireless Rural and Emergency Communications - WRECOM07*, 2007.

[25] M. Musolesi and C. Mascolo, “Mobility Models for Systems Evaluation. A Survey,” in *Middleware for Network Eccentric and Mobile Applications* (B. Garbinato, H. Miranda, and L. Rodrigues, eds.), pp. 43–62, Springer, feb 2009.

[26] A. Ribeiro and R. Sofia, “A Survey on Mobility Models for Wireless Networks,” Tech. Rep. SITI-TR-11-01, SITI, University Lus{ó}fona, feb 2011.

[27] C. Boldrini and A. Passarella, “HCMM: Modelling spatial and temporal properties of human mobility driven by users’ social relationships,” *Computer Communications*, vol. 33, pp. 1056–1074, jun 2010.

[28] Z. Wang, Y. Xu, L. Li, H. Tian, and S. Cui, “Handover Control in Wireless Systems via Asynchronous Multi-User Deep Reinforcement Learning,” pp. 1–11, 2018.

[29] S. Peters, D. P. Pardo, and Q. Zhou, “Mobility Management in ULOOP,” in *User-Centric Networking - Future Perspectives, Springer Lecture Notes in Social Networks*, pp. 311–325, 2014.

[30] R. Sofia, T. Condeixa, and S. Sargento, “Mobility Estimation in the Context of Distributed Mobility Management,” in *User-centric Networking and Services - Future Perspectives*, 2014.

[31] R. C. Sofia, “User-centric Networking: bringing the Home Network to the Core,” in *User-centric Networking and Services - Future Perspectives*, Lecture Notes on Social Networking, ch. Part I, pp. 3–23, Springer, user-centr ed., 2014.

7 Annexes

7.1 Annex I - Full Results

7.1.1 Scenario I, Control Experiment, 11 a.m.

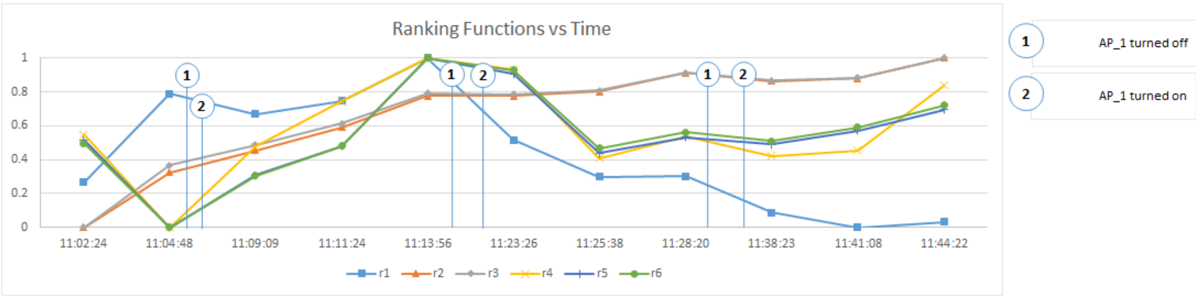


Figure 23: Ranking results over time, scenario I, run II, 11a.m.

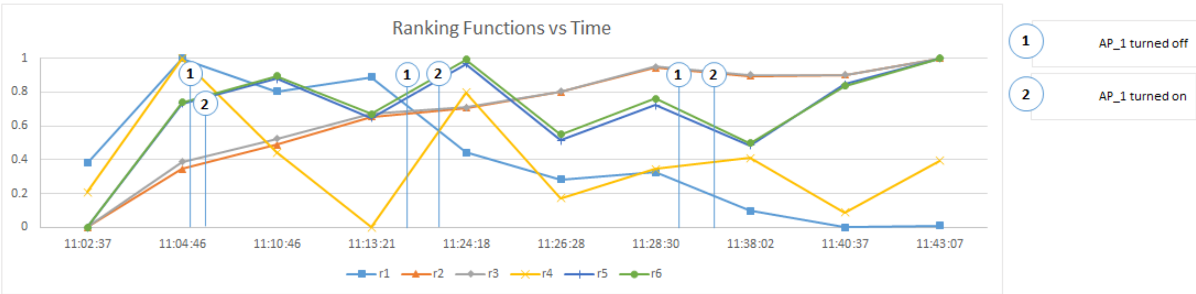


Figure 24: Ranking results over time, scenario I, run III, 11a.m.

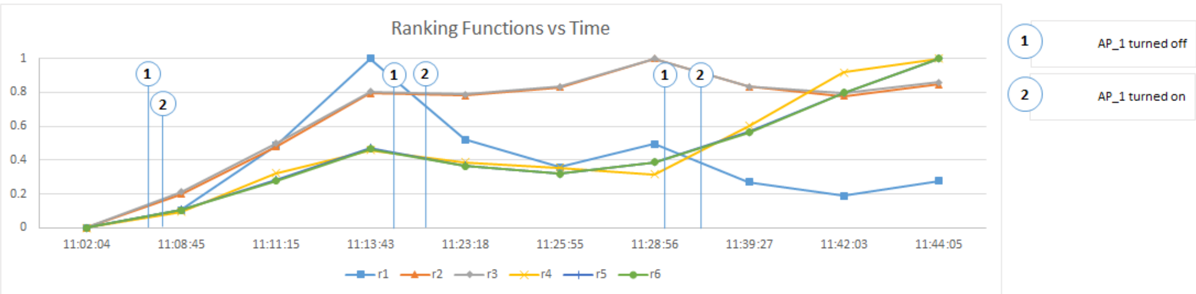


Figure 25: Ranking results over time, scenario I, run IV, 11a.m.

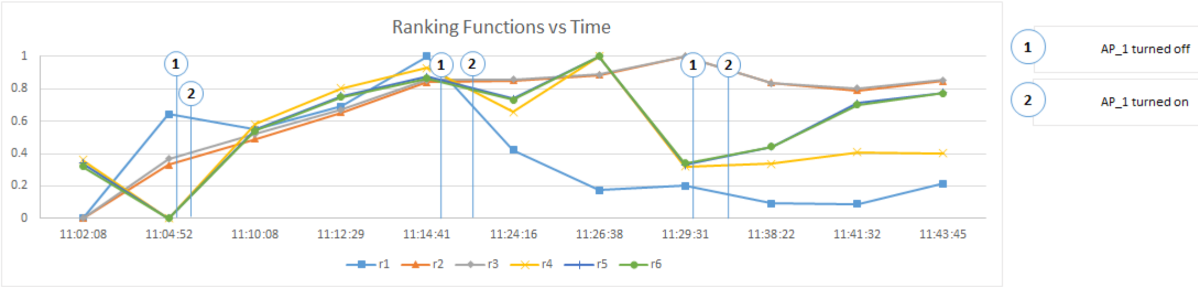


Figure 26: Ranking results over time, scenario I, run V, 11a.m.

7.1.2 Scenario I Control Experiment, 5 p.m

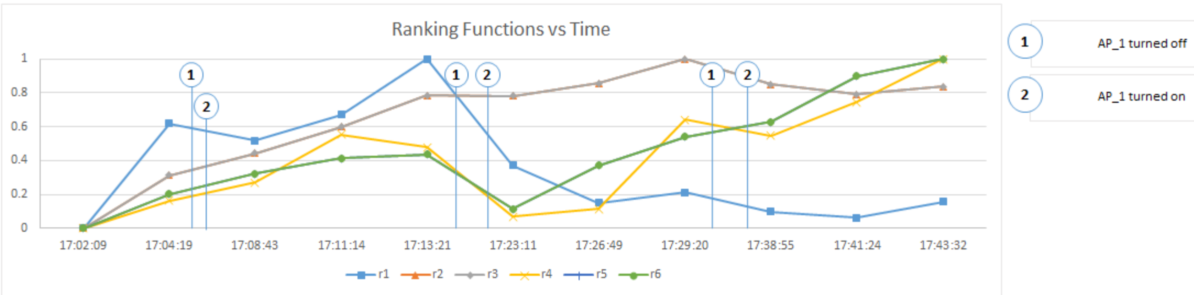


Figure 27: Ranking results over time, scenario I, run II, 5 p.m.

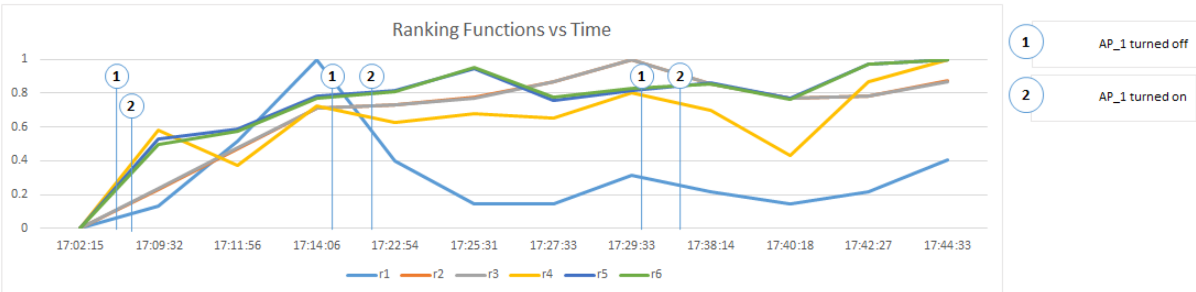


Figure 28: Ranking results over time, scenario I, run III, 5 p.m.

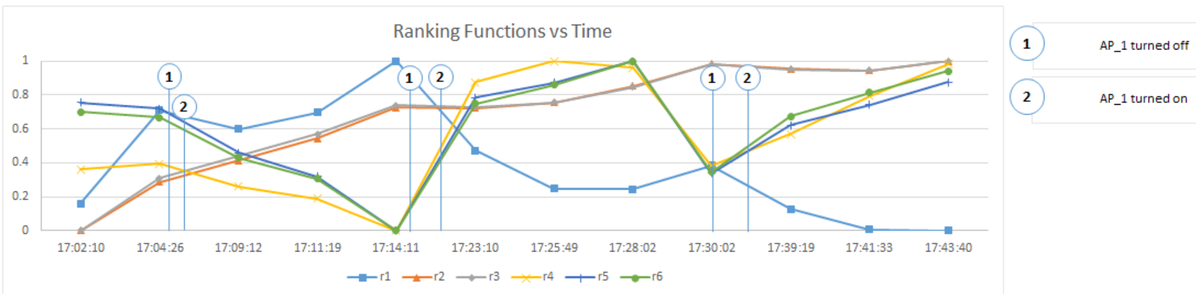


Figure 29: Ranking results over time, scenario I, run IV, 5 p.m.

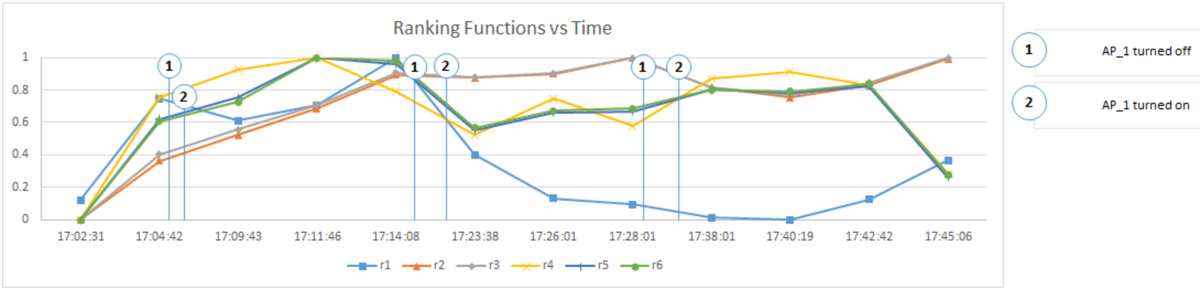


Figure 30: Ranking results over time, scenario I, run V, 5 p.m.

7.1.3 Scenario II - Experiments, 11 a.m.

- **Run II**

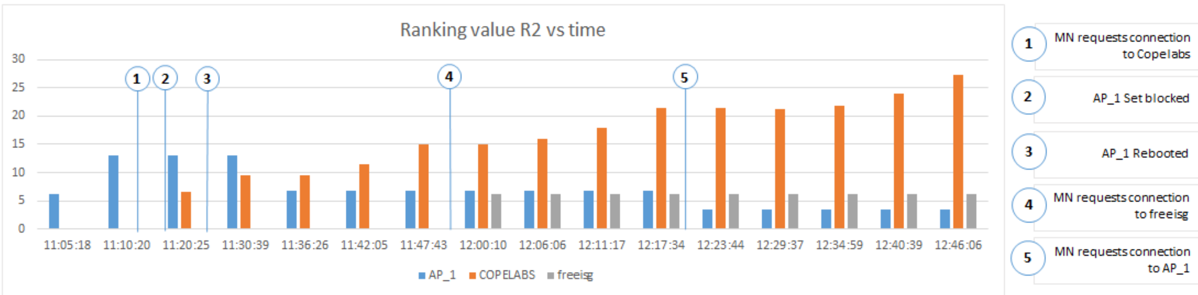


Figure 31: Scenario II, r2 results, perspective of device I, run II.

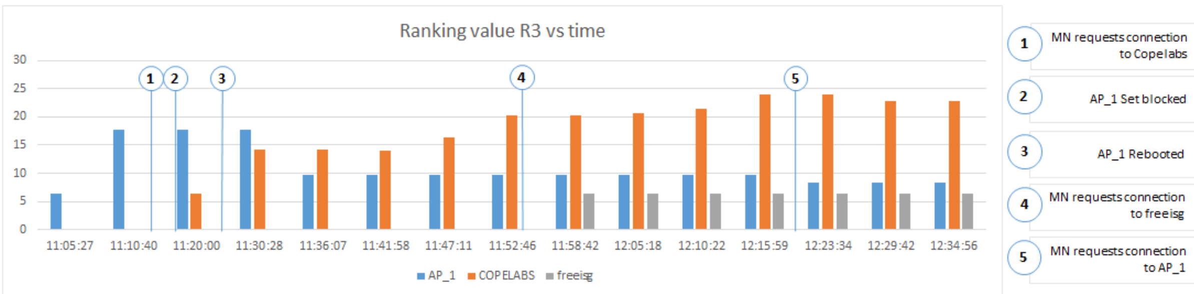


Figure 32: Scenario II, r3 results, perspective of device II, run II.

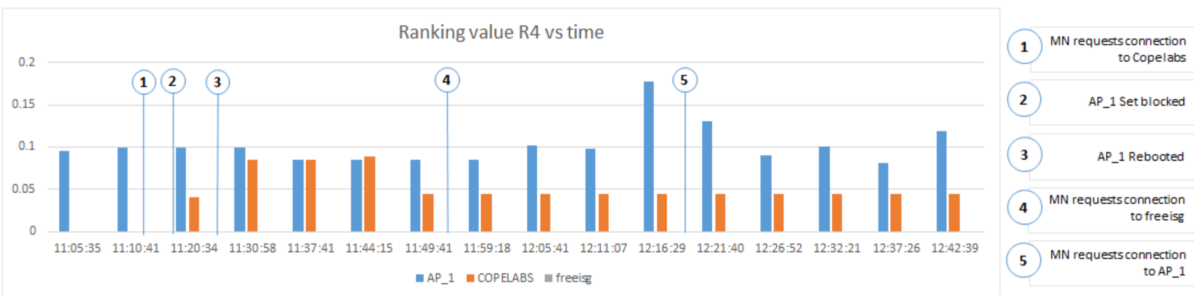


Figure 33: Scenario II, r4 results, perspective of device IV, run II.

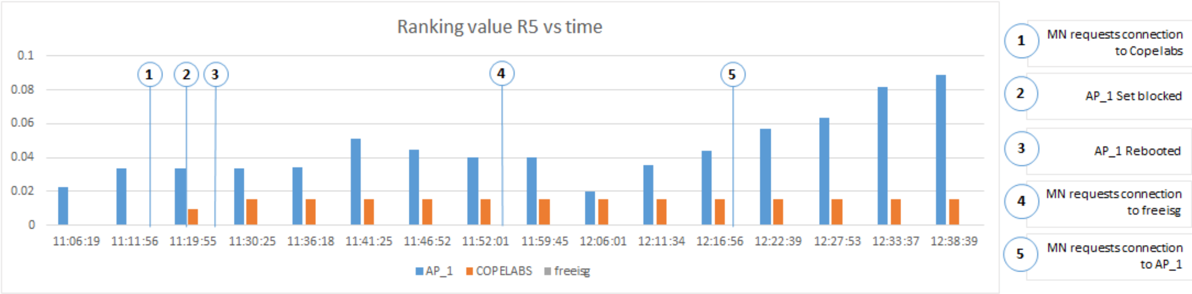


Figure 34: Scenario II, r5 results, perspective of device V, run II.

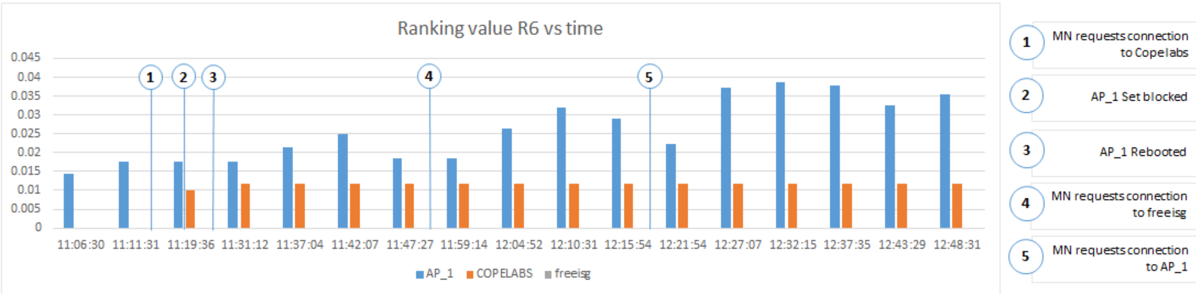


Figure 35: Scenario II, r6 results, perspective of device VII, run II.

- **Run III**

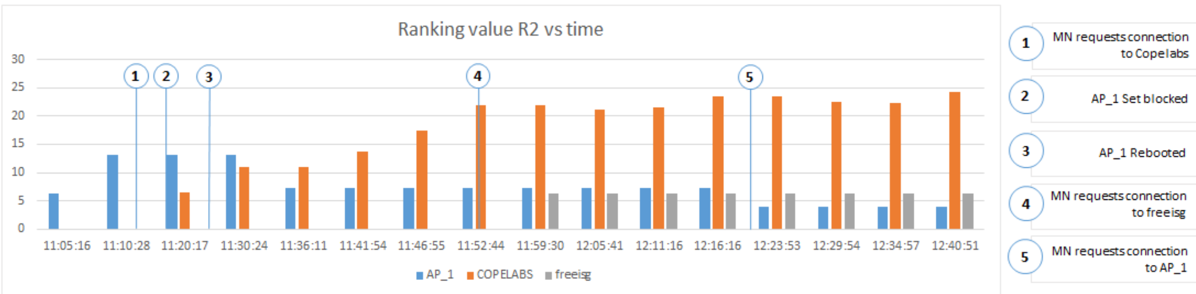


Figure 36: Scenario II, r2 results, perspective of device I, run III.

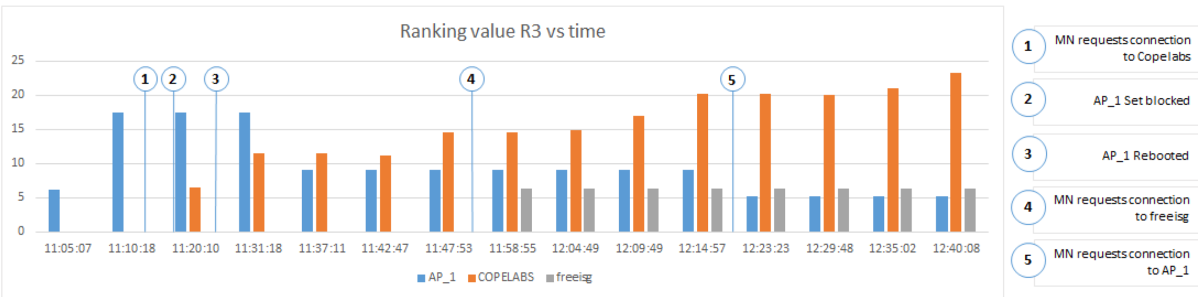


Figure 37: Scenario II, r3 results, perspective of device II, run III.

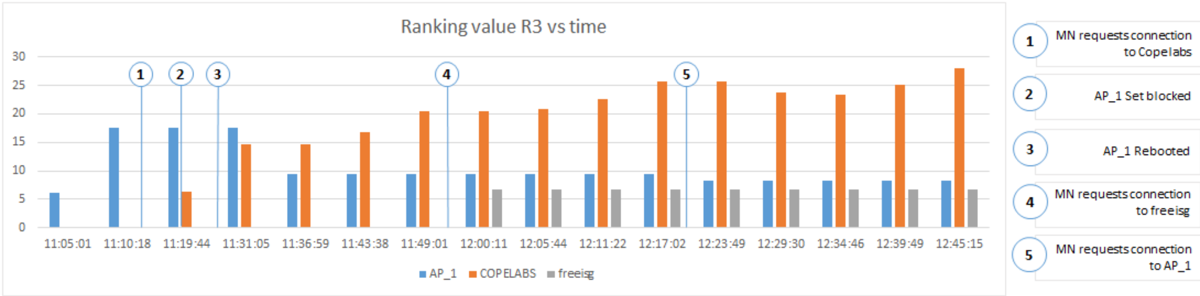


Figure 38: Scenario II, r3 results, perspective of device III, run III.

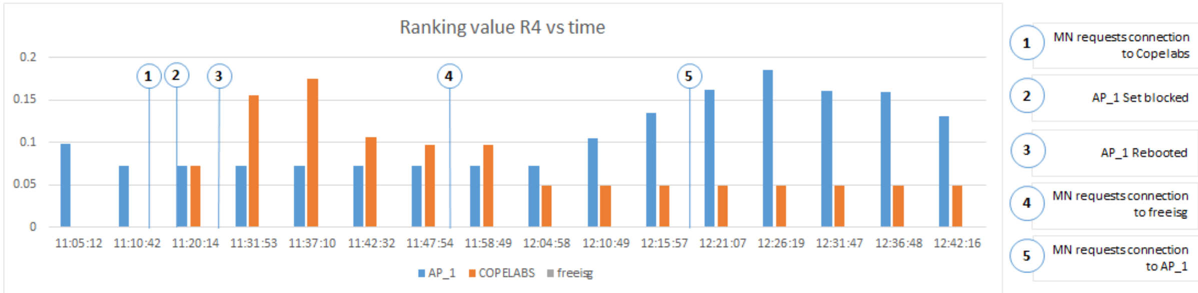


Figure 39: Scenario II, r4 results, perspective of device IV, run III.

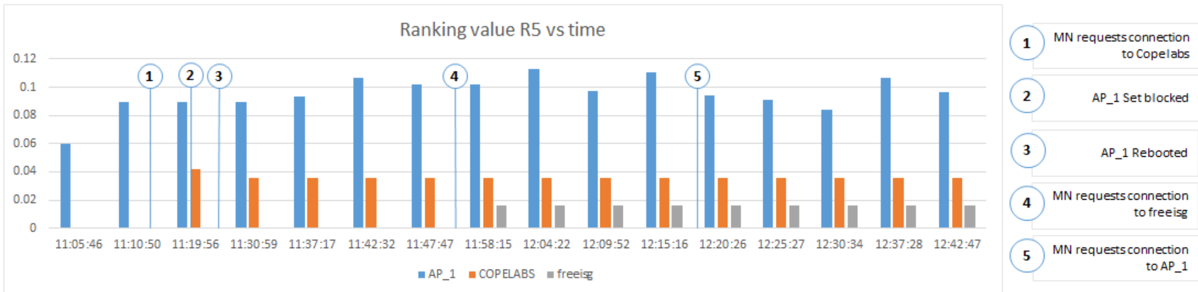


Figure 40: Scenario II, r5 results, perspective of device V, run III.

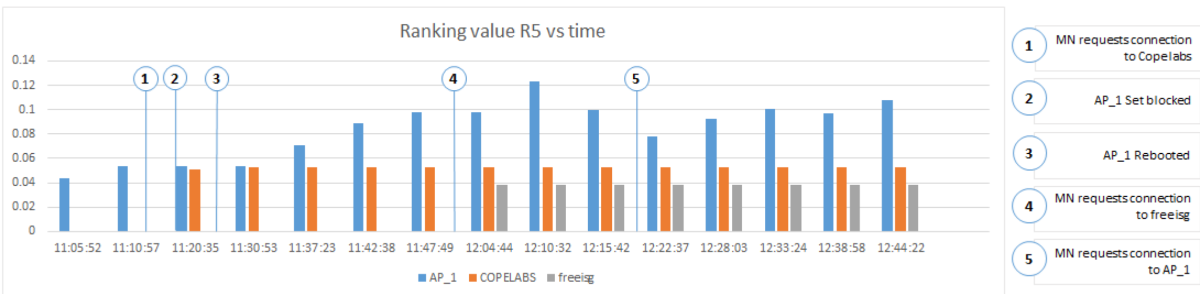


Figure 41: Scenario II, r5 results, perspective of device VI, run III.

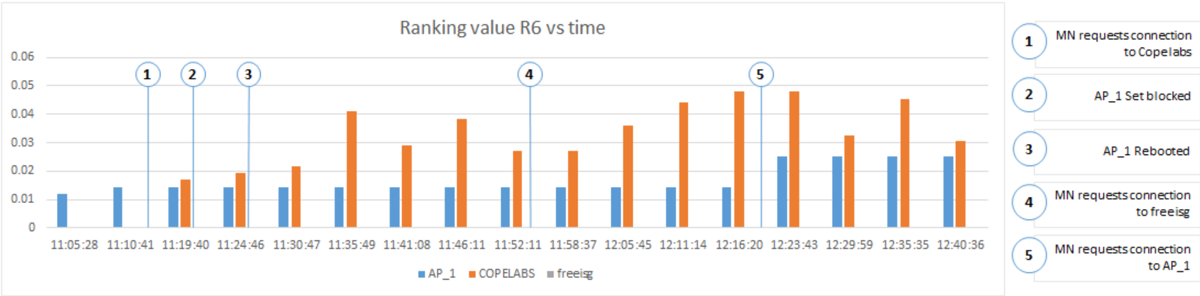


Figure 42: Scenario II, r6 results, perspective of device VII, run III.

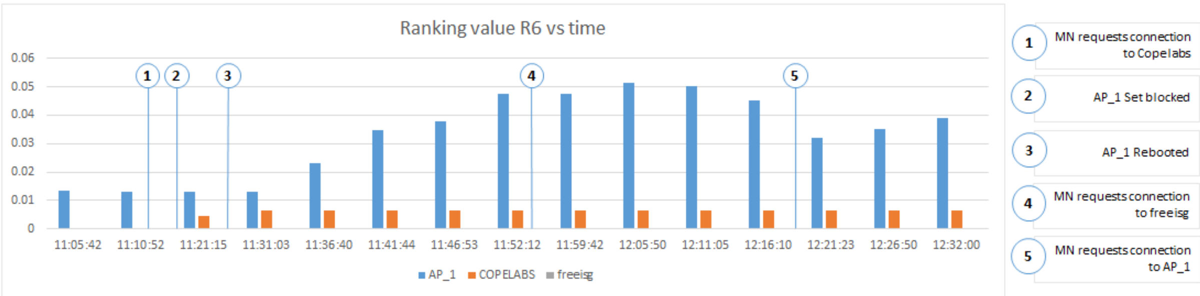


Figure 43: Scenario II, r6 results, perspective of device VIII, run III.

- **Run IV**

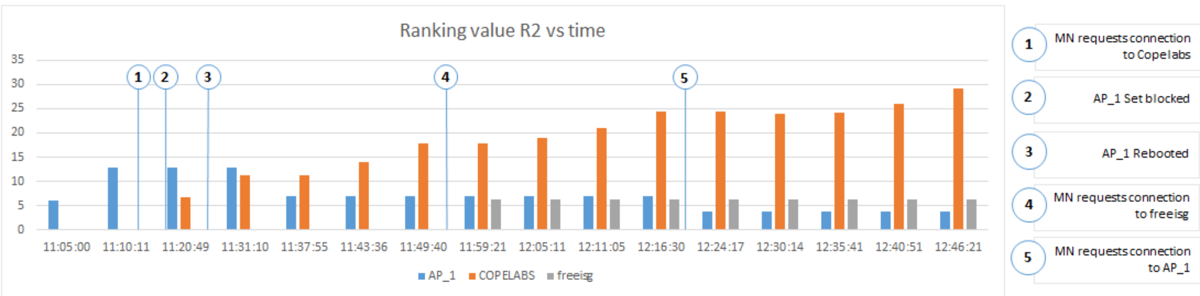


Figure 44: Scenario II, r2 results, perspective of device I, run IV.

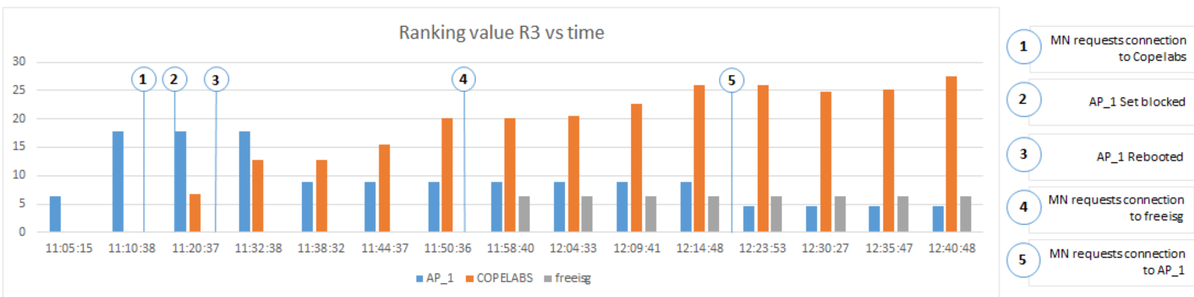


Figure 45: Scenario II, r3 results, perspective of device II, run IV.

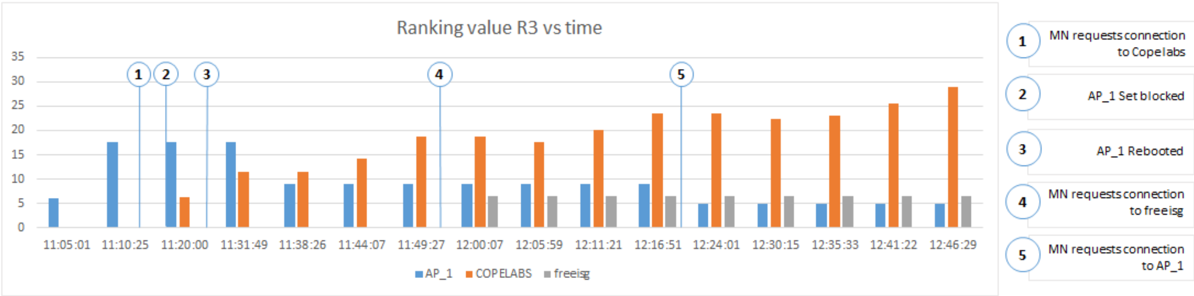


Figure 46: Scenario II, r3 results, perspective of device III, run IV.

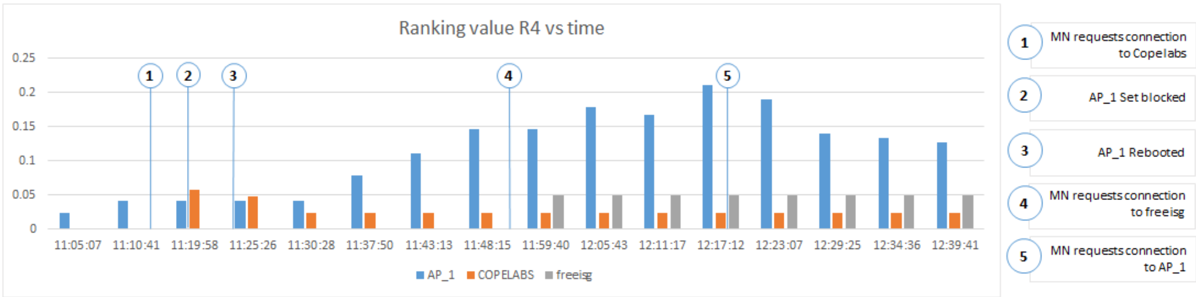


Figure 47: Scenario II, r4 results, perspective of device IV, run IV.

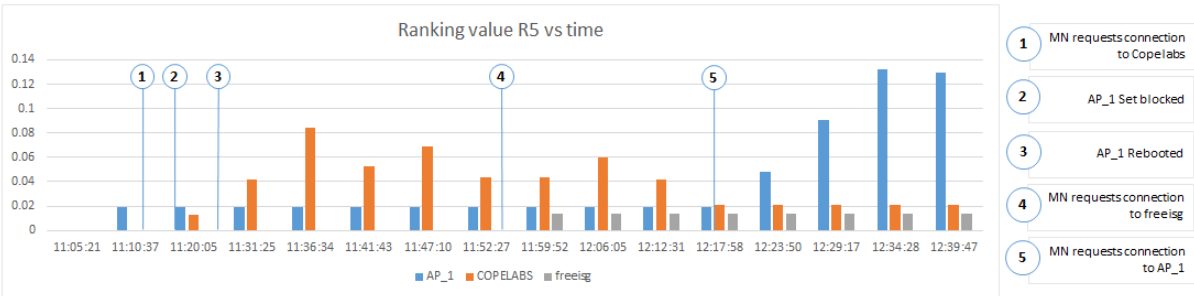


Figure 48: Scenario II, r5 results, perspective of device V, run IV.

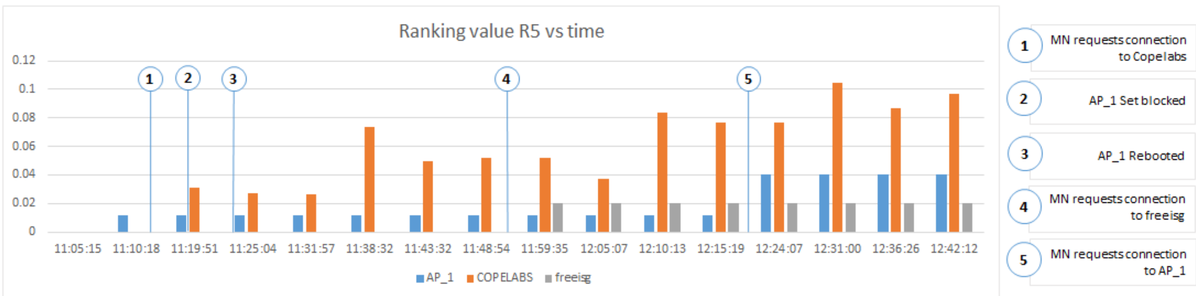


Figure 49: Scenario II, r5 results, perspective of device VI, run IV.

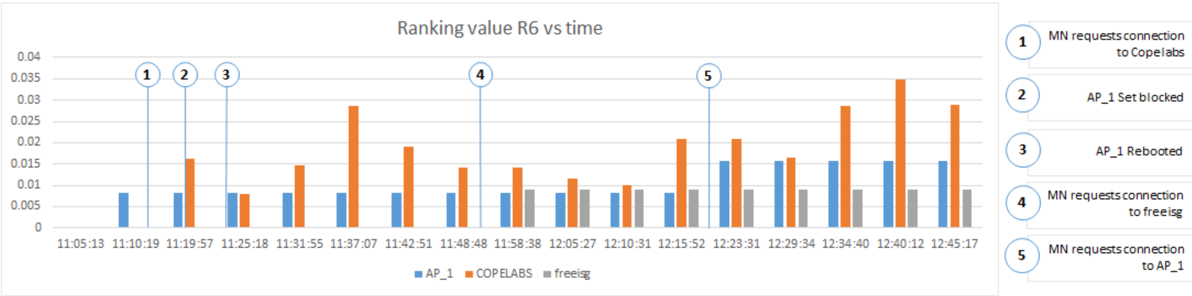


Figure 50: Scenario II, r6 results, perspective of device VII, run IV.

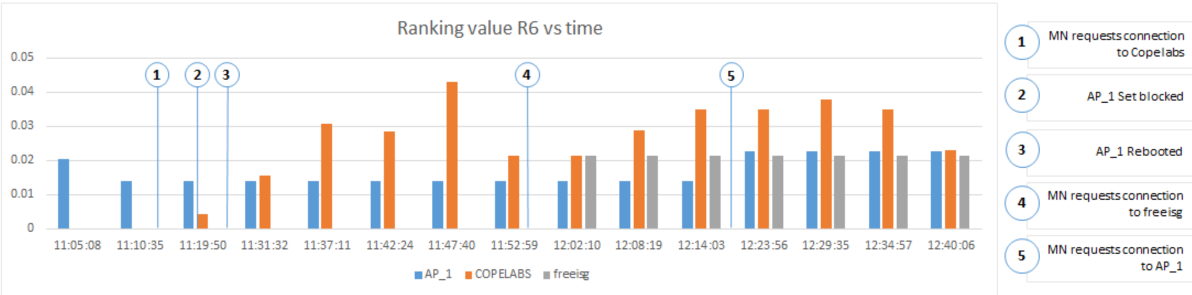


Figure 51: Scenario II, r6 results, perspective of device VIII, run IV.

- **Run V**

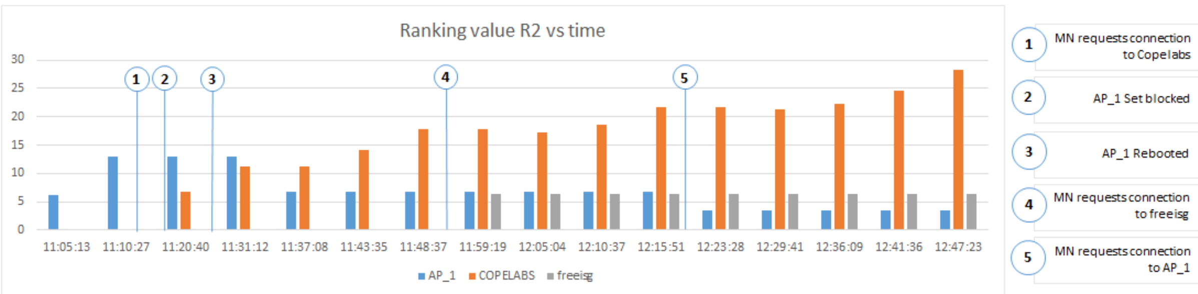


Figure 52: Scenario II, r2 results, perspective of device I, run V.

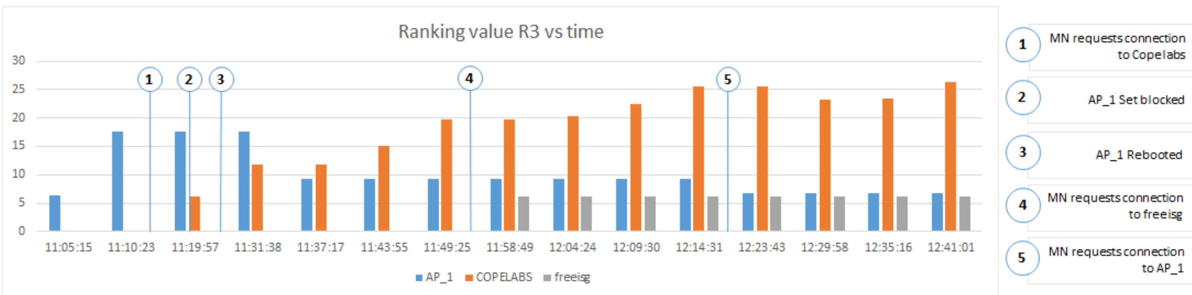


Figure 53: Scenario II, r3 results, perspective of device II, run V.

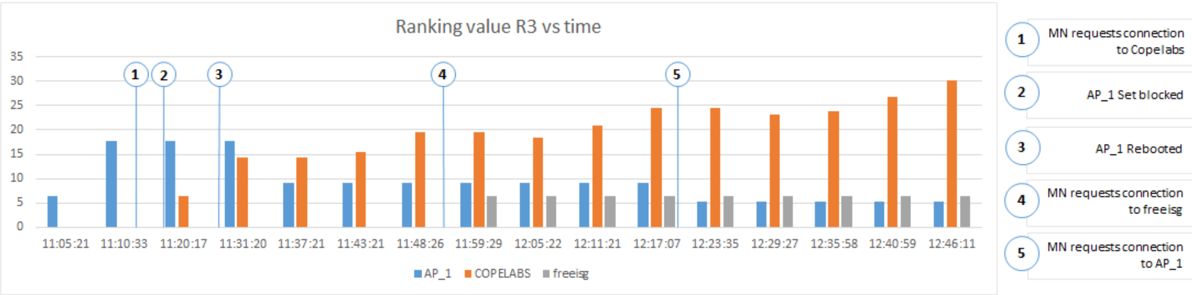


Figure 54: Scenario II, r3 results, perspective of device III, run V.

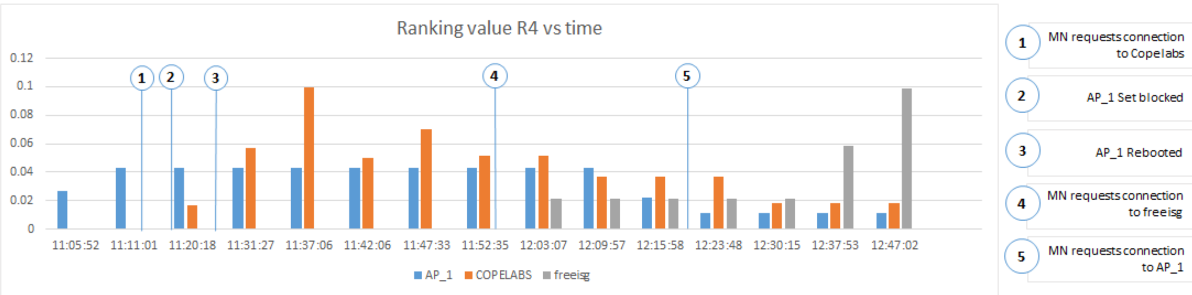


Figure 55: Scenario II, r4 results, perspective of device IV, run V.

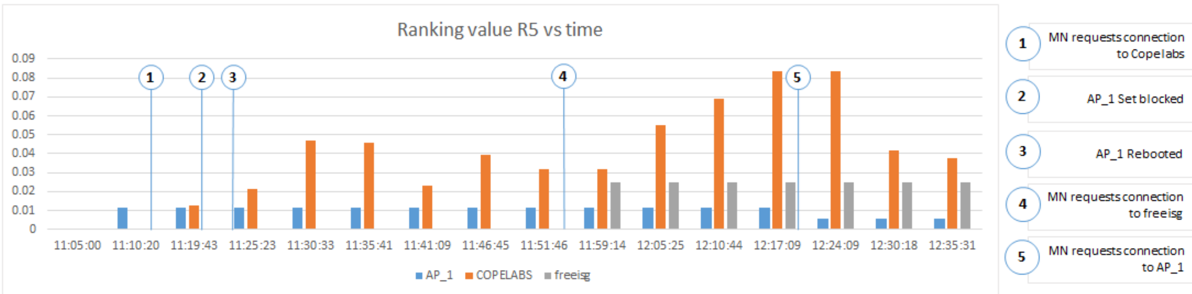


Figure 56: Scenario II, r5 results, perspective of device V, run V.

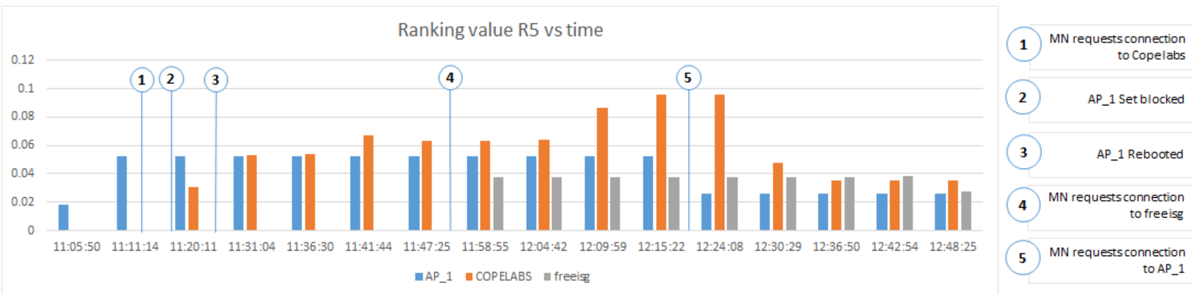


Figure 57: Scenario II, r5 results, perspective of device VI, run V.

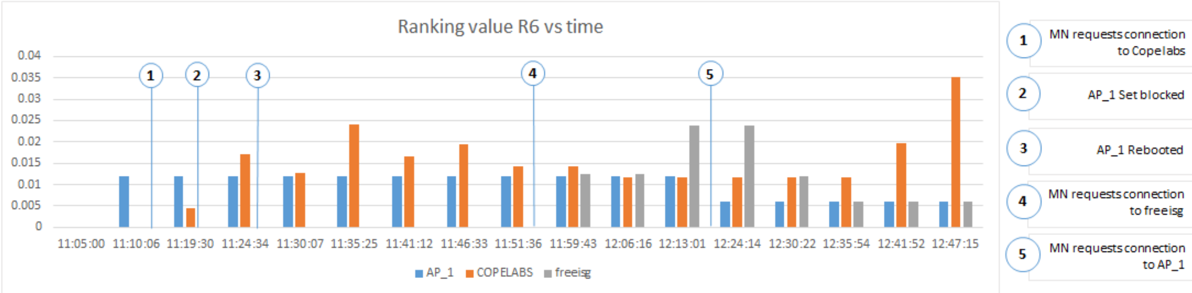


Figure 58: Scenario II, r6 results, perspective of device VII, run V.

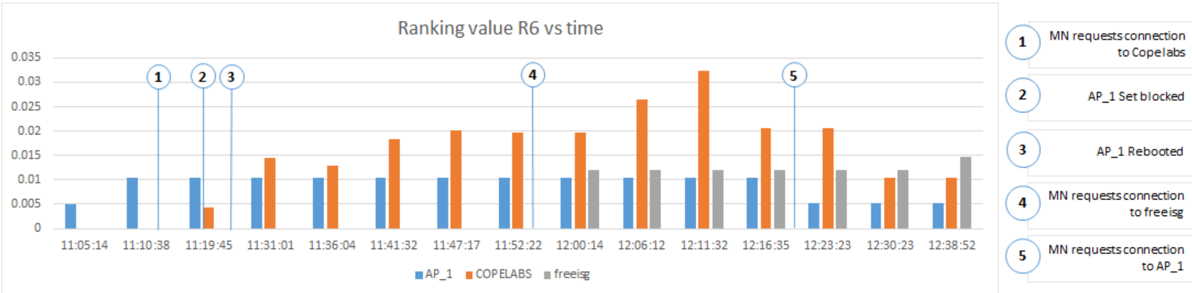


Figure 59: Scenario II, r6 results, perspective of device VIII, run V.

7.1.4 Scenario II test results time 5pm

- **Run II**

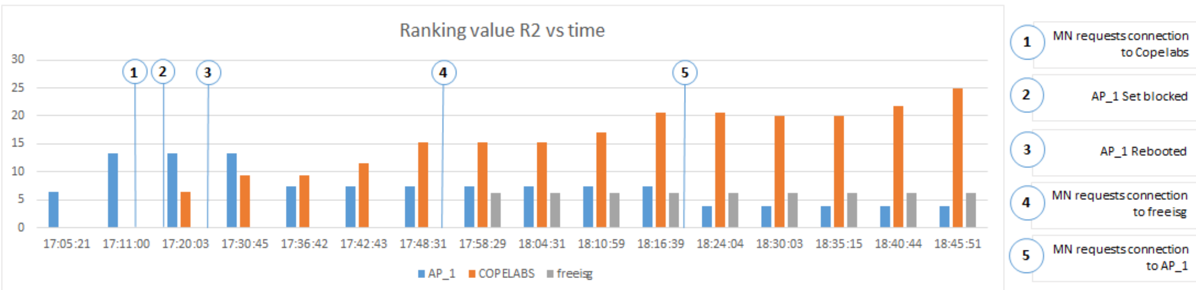


Figure 60: Scenario II, r2 results, perspective of device I, run II.

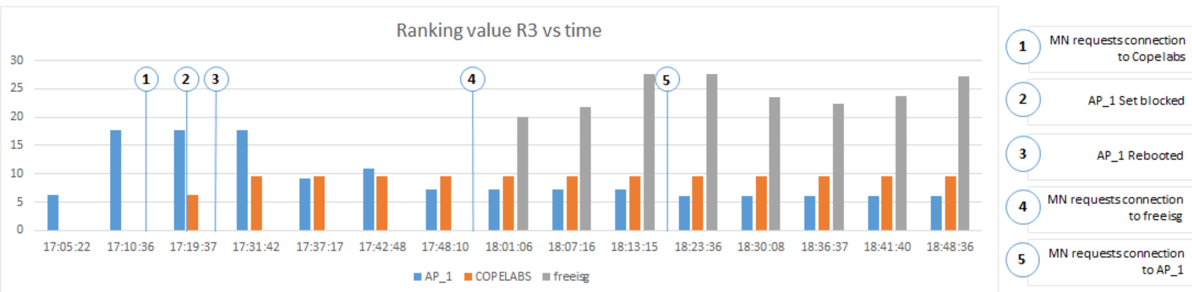


Figure 61: Scenario II, r3 results, perspective of device II, run II.

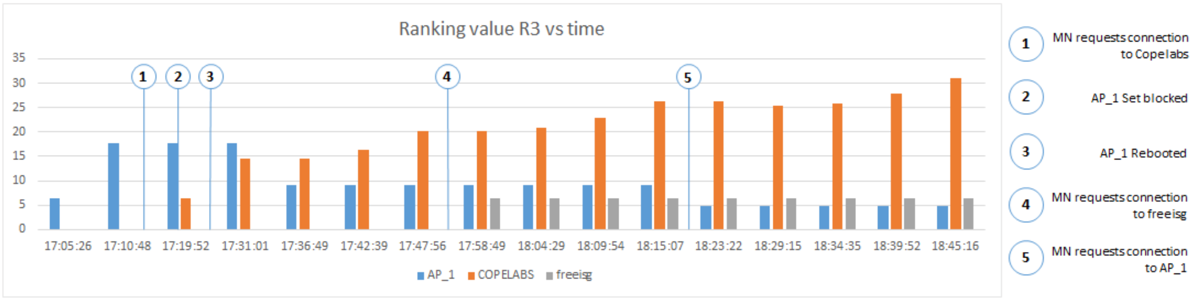


Figure 62: Scenario II, r3 results, perspective of device III, run II.

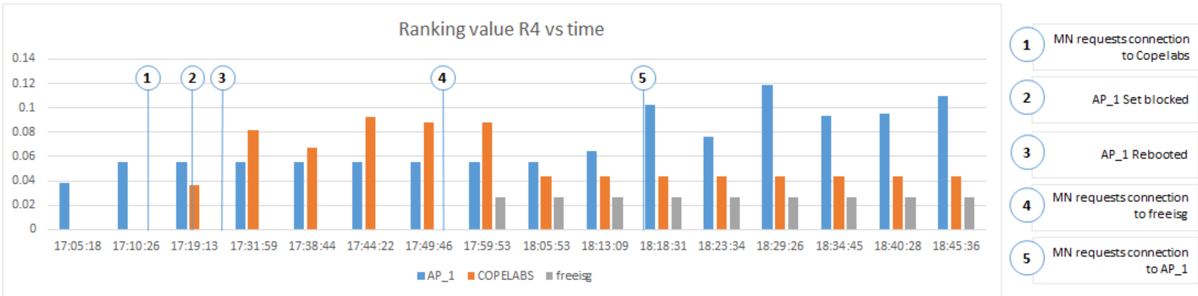


Figure 63: Scenario II, r4 results, perspective of device IV, run II.

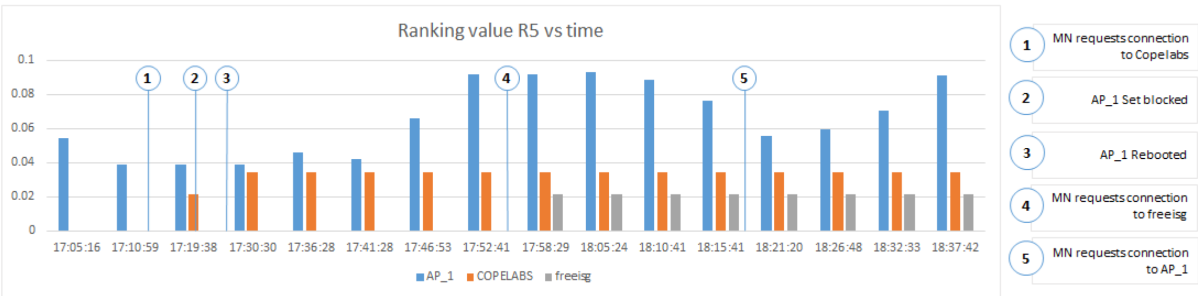


Figure 64: Scenario II, r5 results, perspective of device V, run V.

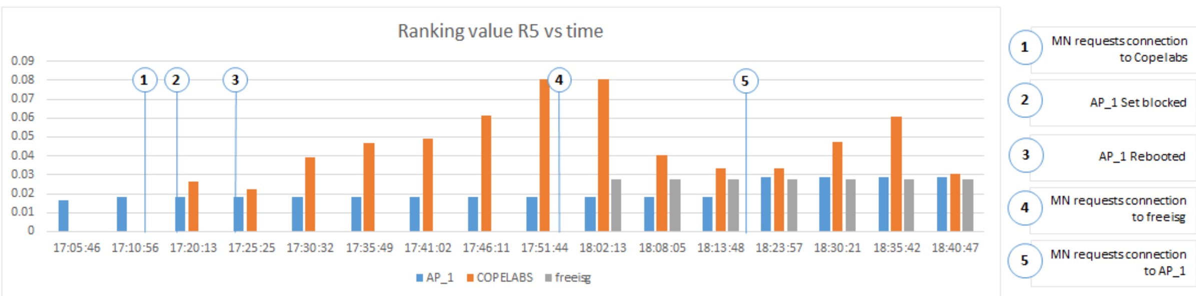


Figure 65: Scenario II, r5 results, perspective of device VI, run II.

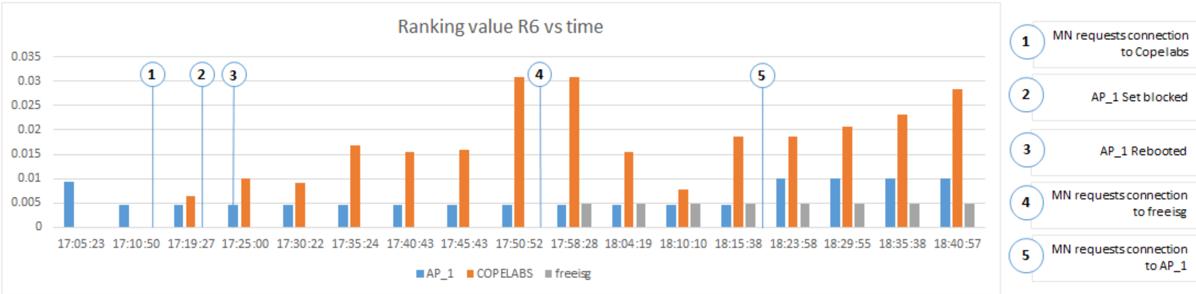


Figure 66: Scenario II, r6 results, perspective of device VII, run II.

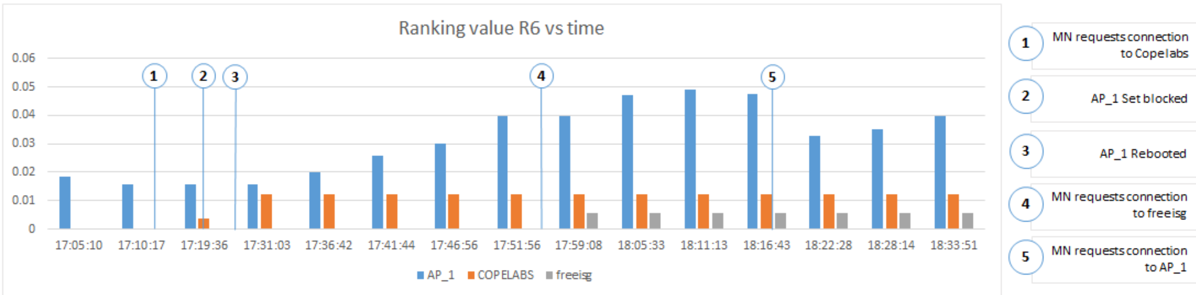
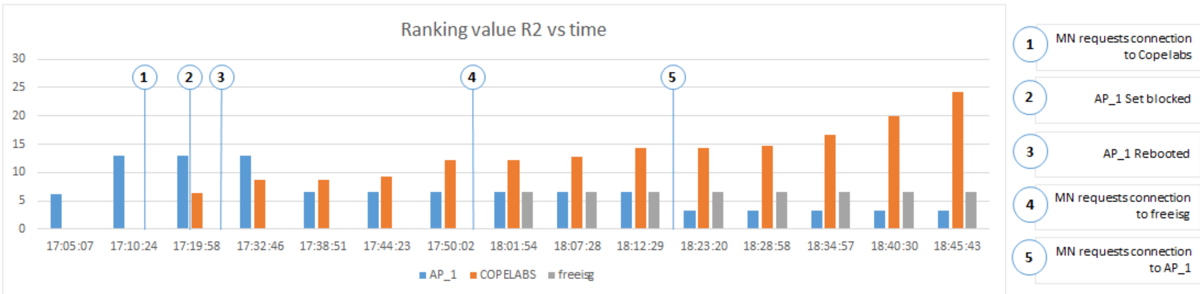


Figure 67: Scenario II, r6 results, perspective of device VIII, run II.

- **Test III**



S

Figure 68: Scenario II, r2 results, perspective of device I, run III.

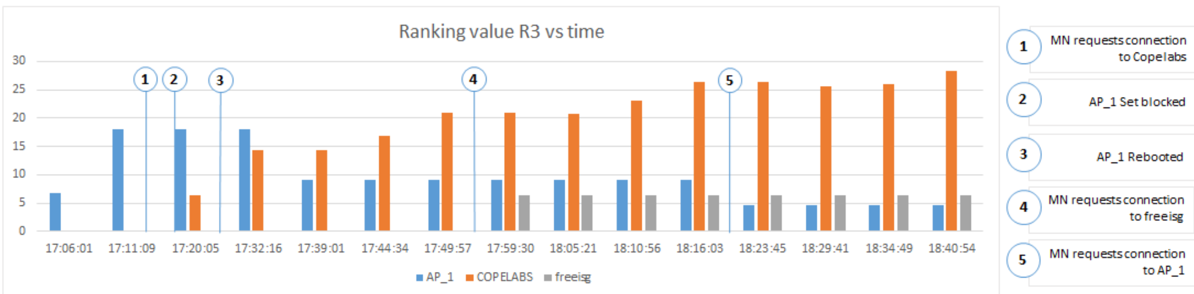


Figure 69: Scenario II, r3 results, perspective of device II, run III.

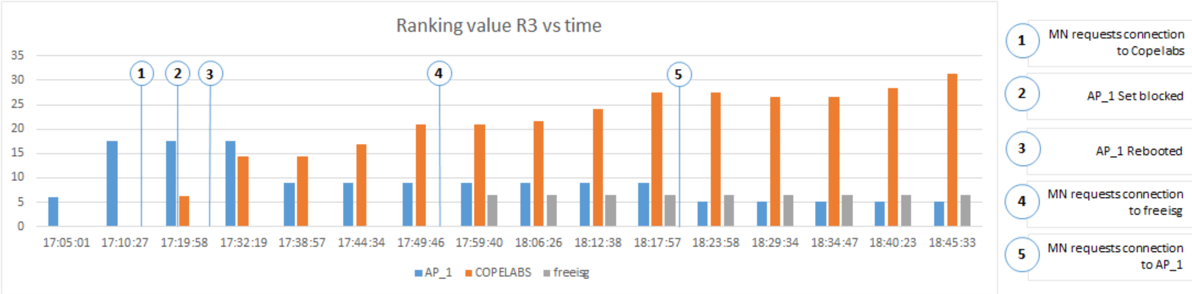


Figure 70: Scenario II, r3 results, perspective of device III, runIII.

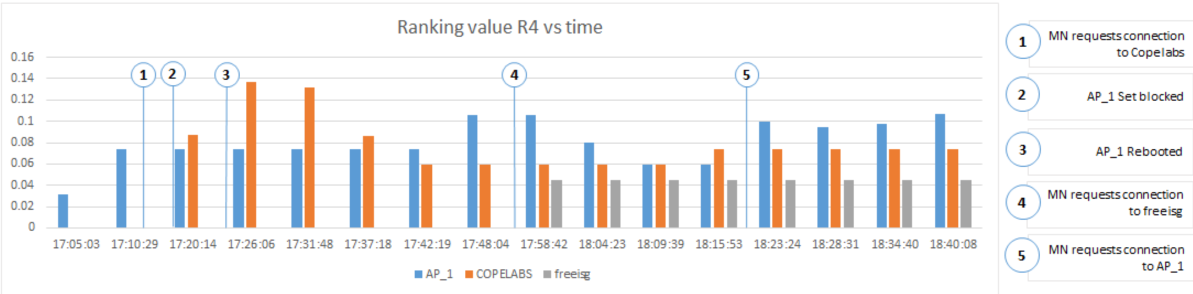


Figure 71: Scenario II, r4 results, perspective of device IV, run III.

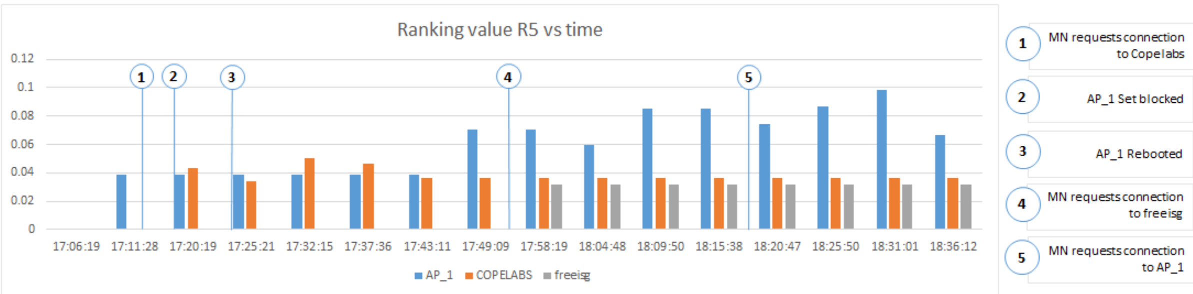


Figure 72: Scenario II, r5 results, perspective of device V, run III.

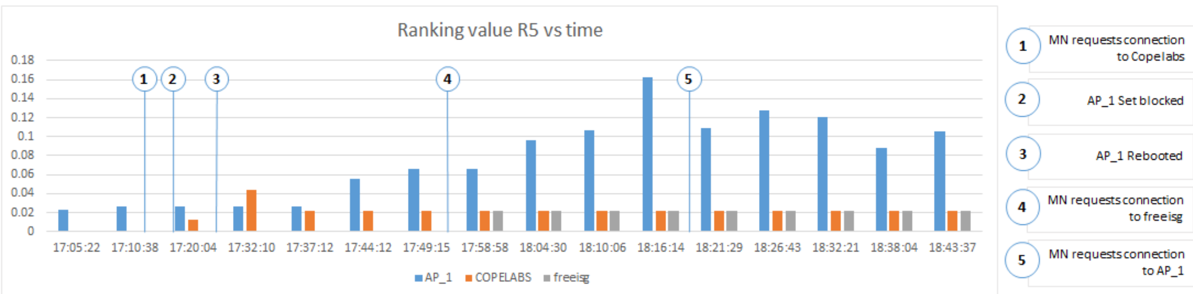


Figure 73: Scenario II, r5 results, perspective of device VI, run III.

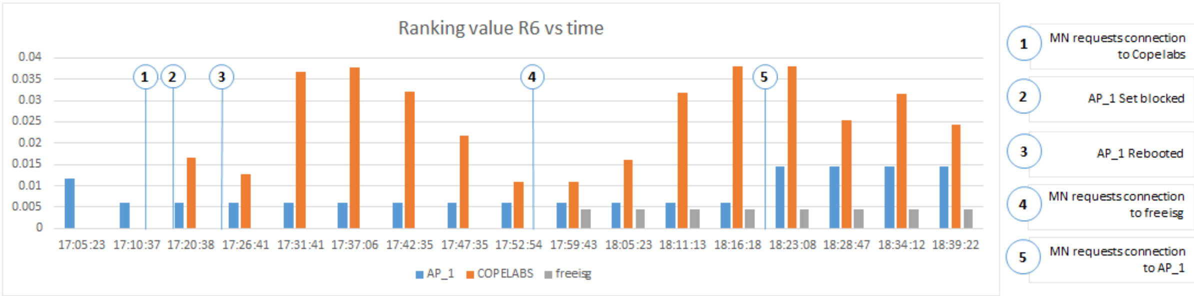


Figure 74: Scenario II, r6 results, perspective of device VII, run III.

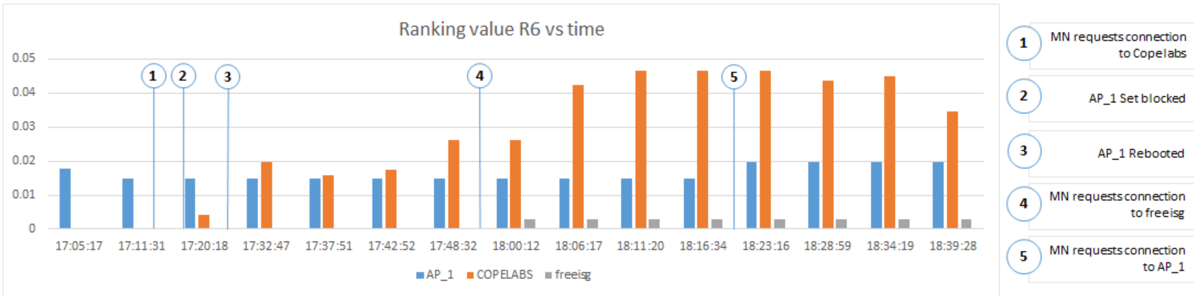


Figure 75: Scenario II, r6 results, perspective of device VIII, run III.

- **Test IV**

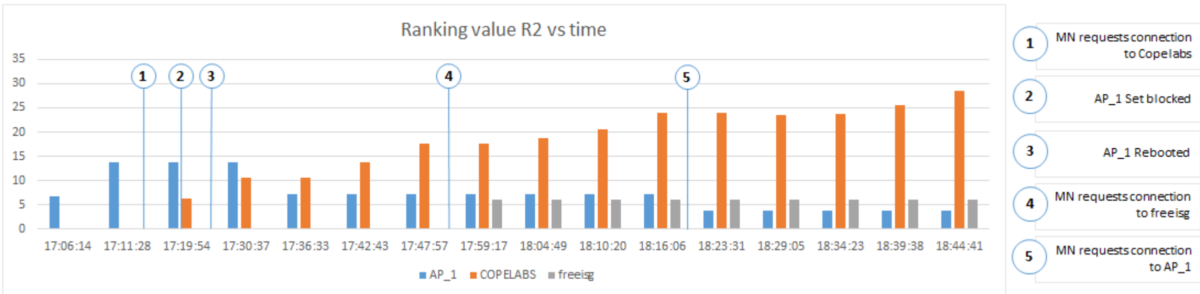


Figure 76: Scenario II, r2 results, perspective of device I, run IV.

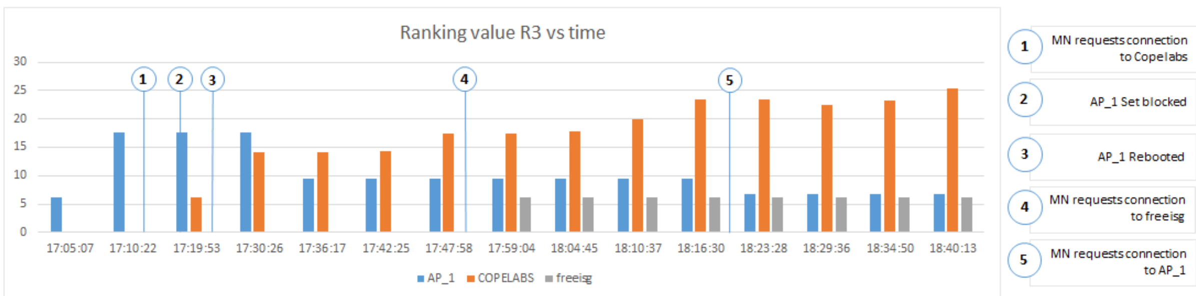


Figure 77: Scenario II, r3 results, perspective of device II, run IV.

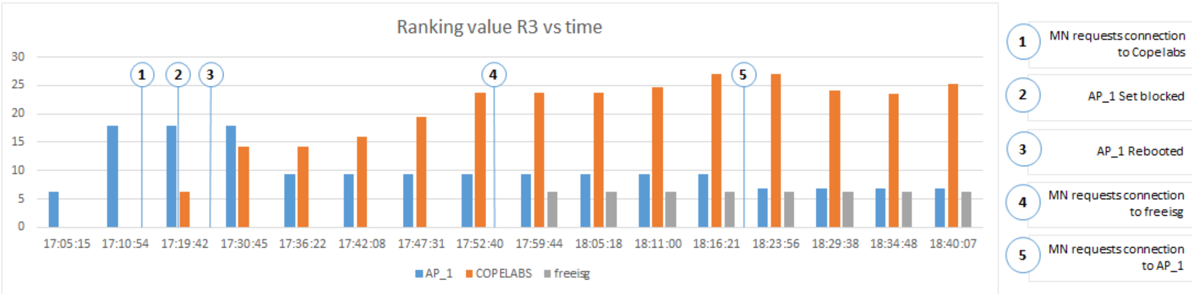


Figure 78: Scenario II, r3 results, perspective of device III, run IV.

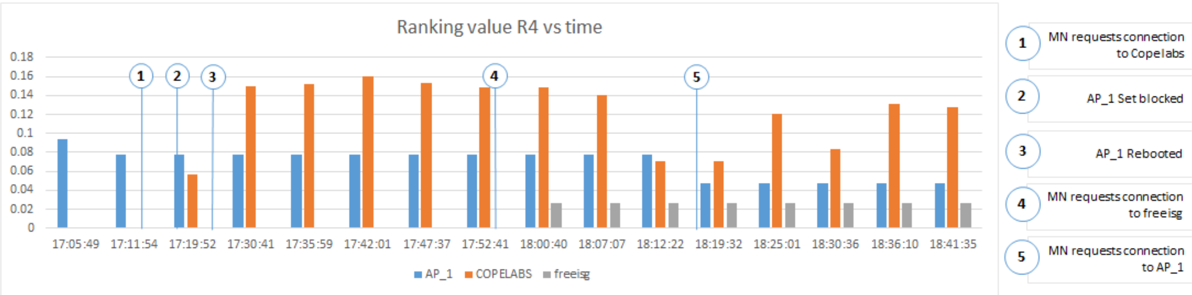


Figure 79: Scenario II, r4 results, perspective of device IV, run IV.

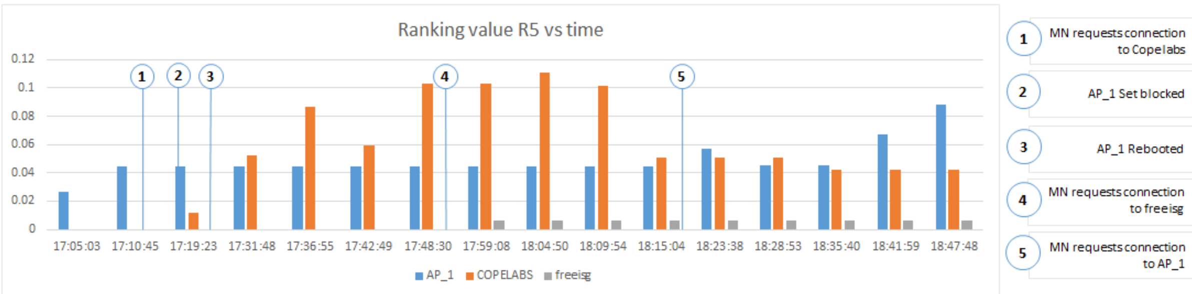


Figure 80: Scenario II, r5 results, perspective of device V, run IV.

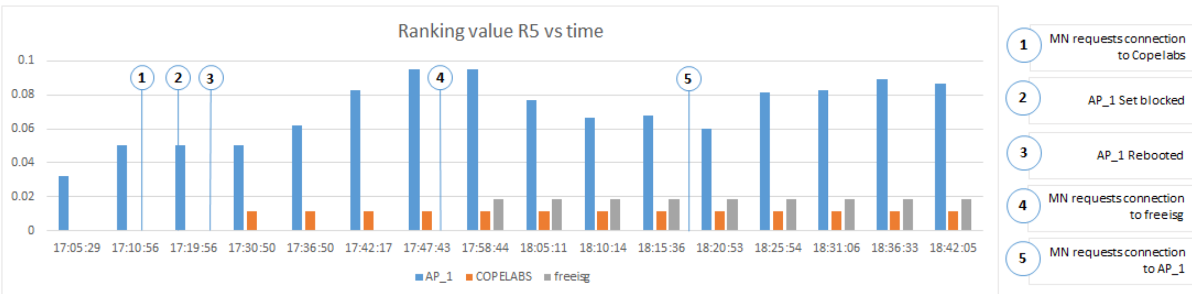


Figure 81: Scenario II, r5 results, perspective of device VI, run IV.

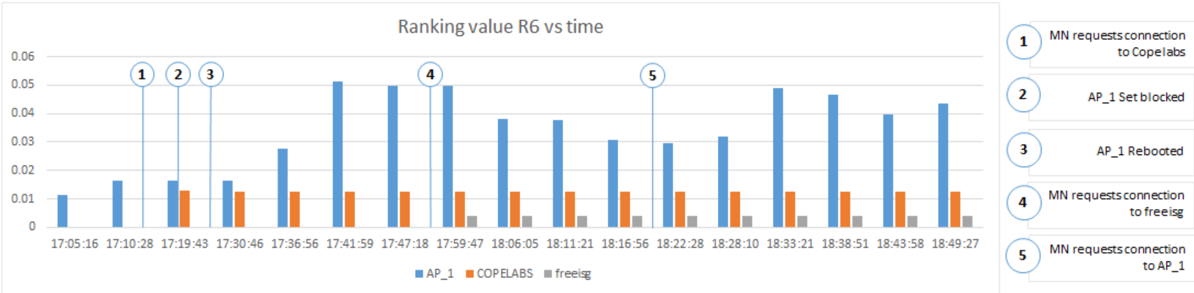


Figure 82: Scenario II, r6 results, perspective of device VII, run IV.

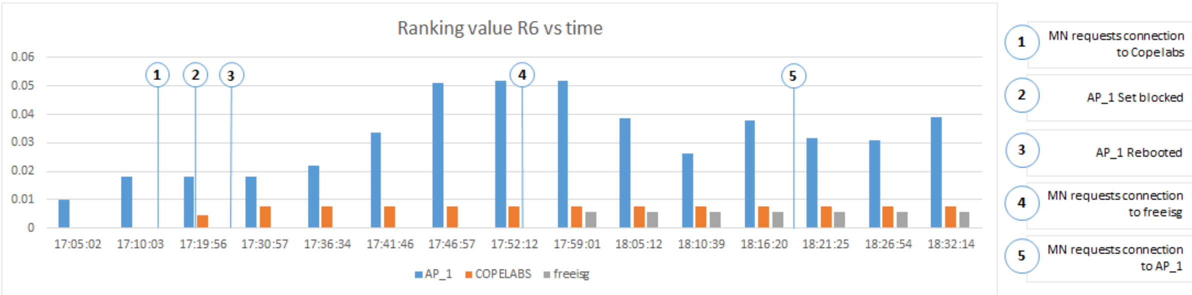


Figure 83: Scenario II, r6 results, perspective of device VIII, run IV.

- **Test V**

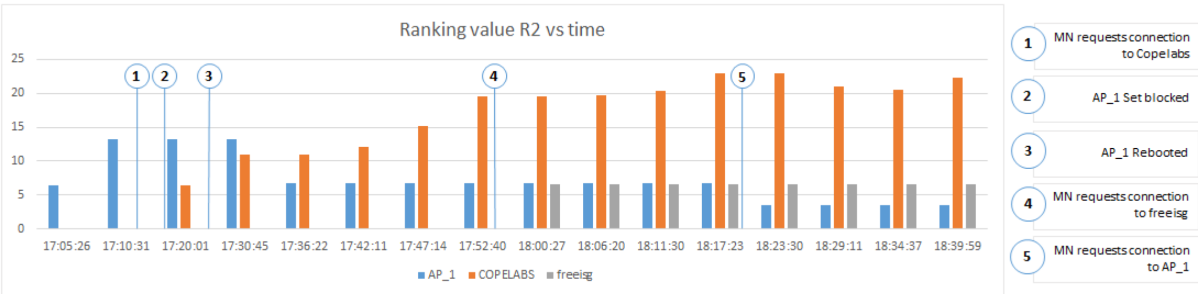


Figure 84: Scenario II, r2 results, perspective of device I, run V.

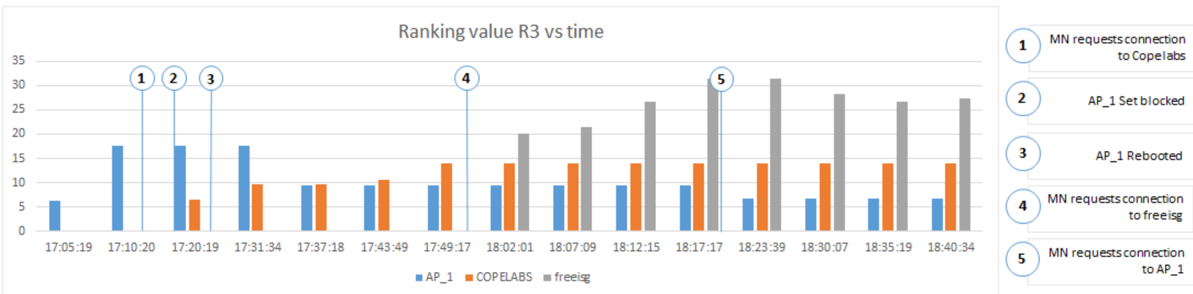


Figure 85: Scenario II, r3 results, perspective of device II, run V.

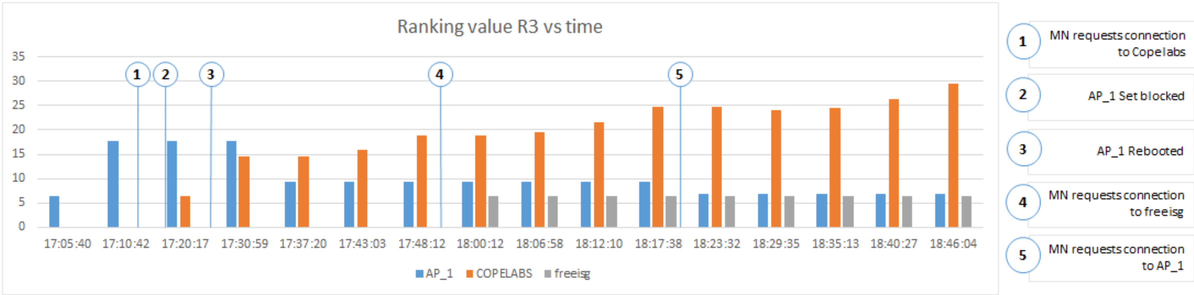


Figure 86: Scenario II, r3 results, perspective of device III, run V.

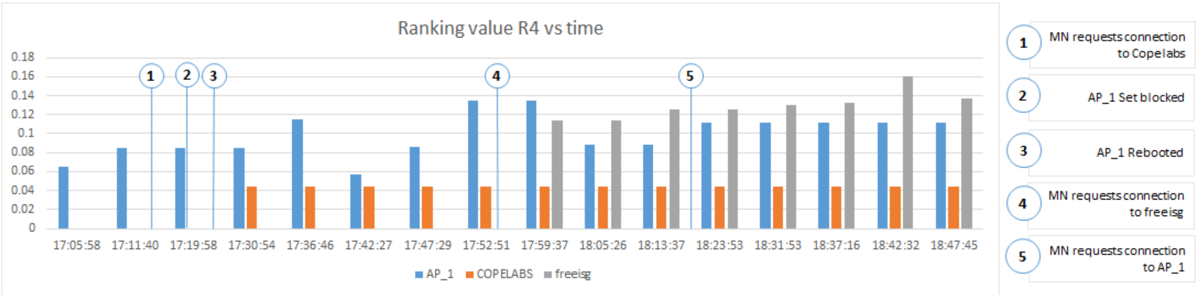


Figure 87: Scenario II, r4 results, perspective of device IV, run V.

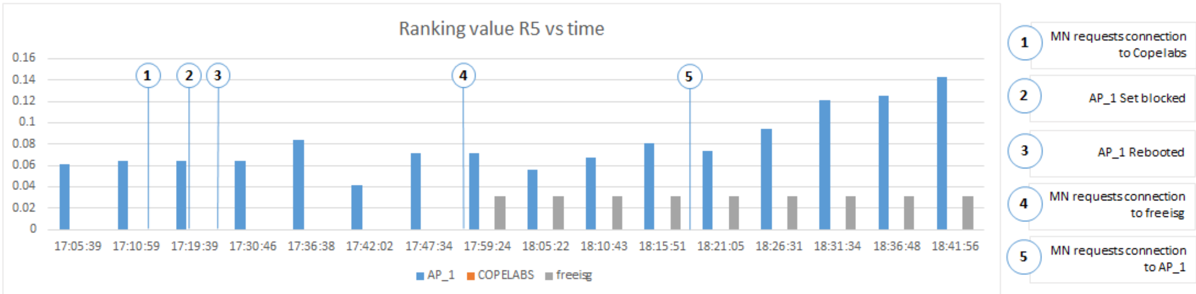


Figure 88: Scenario II, r5 results, perspective of device V, run V.

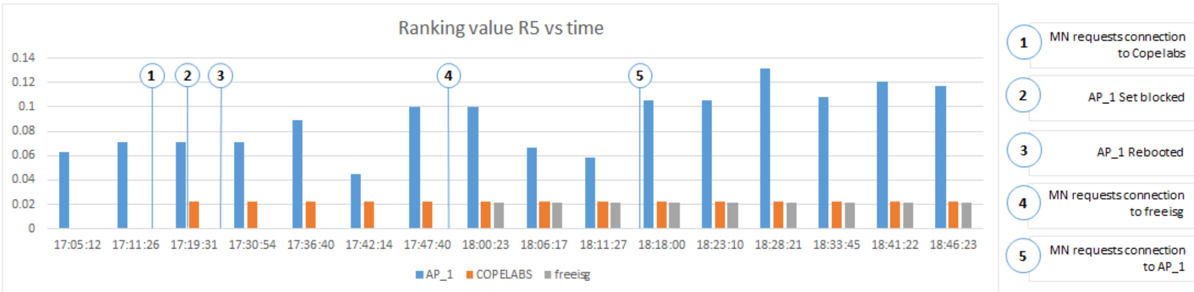


Figure 89: Scenario II, r5 results, perspective of device VI, run V.

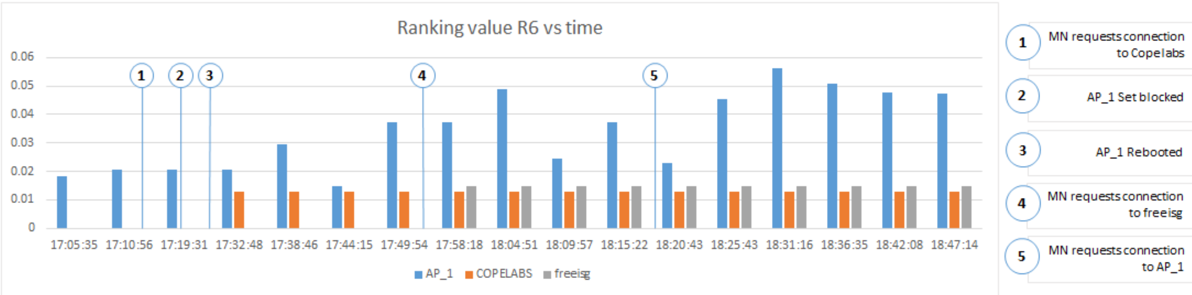


Figure 90: Scenario II, r6 results, perspective of device VII, run V.

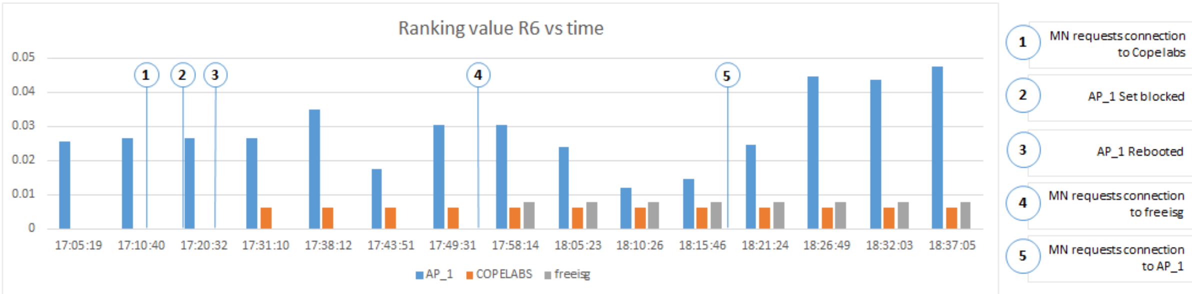


Figure 91: Scenario II, r6 results, perspective of device VIII, run V.

7.2 Annex II - Code Documentation

JavaDoc

Note: The following document contains its owns indexs of tables and figures.

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Package cs

Packages

- package **usense**

5.2 Package cs.usense

Packages

- package **pipelines**

5.3 Package cs.usense.pipelines

Packages

- package **mobility**

5.4 Package cs.usense.pipelines.mobility

Packages

- package **fragments**
- package **functions**
- package **helpers**
- package **interfaces**
- package **mobilitytracker**
- package **models**
- package **tasks**
- package **utils**

5.5 Package cs.usense.pipelines.mobility.fragments

Classes

- class **AttractivenessDialogFragment**

5.6 Package cs.usense.pipelines.mobility.functions

Classes

- class **Functions**
- class **ProbingFunctionsManager**

5.7 Package cs.usense.pipelines.mobility.helpers

Classes

- class **MTrackerDataSource**
- class **MTrackerSQLiteHelper**

5.8 Package cs.usense.pipelines.mobility.interfaces

Classes

- interface **DataBaseChangeListener**
- interface **WifiChangeListener**

5.8.1 Detailed Description

ULOOP Mobility tracking plugin: Mtracker

Mtracker is an Android app that collects information concerning visited APs It computes a rank and then estimates a potential handover - time and target AP v1.0 - pre-prototype, D3.3, July 2012 v2.0 - prototype on September 2012 - D3.6 v3.0 - prototype on June 2013

Author

Jonnahtan Saltarin
Rute Sofia
Christian da Silva Pereira
Luis Amaral Lopes

Version

3.0

5.9 Package cs.usense.pipelines.mobility.mobilitytracker

Classes

- class **MTrackerApplication**
- class **MTrackerService**
- class **MTrackerWifiManager**

5.9.1 Detailed Description

ULOOP Mobility tracking plugin: Mtracker

Mtracker is an Android app that collects information concerning visited APs It computes a probingFunctionsManager and then estimates a potential handover - time and target AP v1.0 - pre-prototype, D3.3, July 2012 v2.0 - prototype on September 2012 - D3.6 v3.0 - prototype on June 2013

5.10 Package cs.usense.pipelines.mobility.models

Classes

- class **MTrackerAP**
- class **MTrackerVisit**

5.11 Package cs.usense.pipelines.mobility.tasks

Classes

- class **ConnectionTask**
- class **DownloadTask**
- class **PeerList**
- class **PingNetworkTask**
- class **TxtRecord**
- class **WifiScan**

5.12 Package cs.usense.pipelines.mobility.utils

Classes

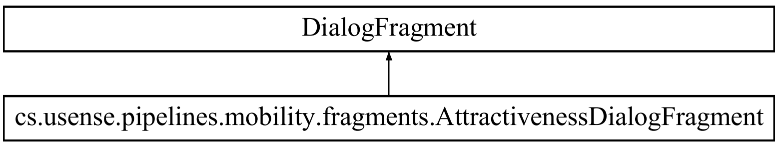
- class **Utils**

Chapter 6

Class Documentation

6.1 cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment:



Classes

- interface **AttractivenessDialogListener**

Public Member Functions

- **AttractivenessDialogFragment** (**MTrackerAP** ap)
- View **onCreateView** (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
- void **onAttach** (Activity activity)

6.1.1 Detailed Description

Created by copelabs on 08/01/2018.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 AttractivenessDialogFragment()

```
cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment.AttractivenessDialog←
Fragment (
              MTrackerAP ap )
```

6.1.3 Member Function Documentation

6.1.3.1 onAttach()

```
void cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment.onAttach (
             Activity activity )
```

6.1.3.2 onCreateView()

```
View cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment.onCreateView (
             LayoutInflater inflater,
             ViewGroup container,
             Bundle savedInstanceState )
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/fragments/ **AttractivenessDialogFragment.java**

6.2 cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment.Attractiveness←DialogListener Interface Reference

Public Member Functions

- void **onUpdateAP** (**MTrackerAP** ap)
- void **connectToAP** (**MTrackerAP** ap)

6.2.1 Member Function Documentation

6.2.1.1 connectToAP()

```
void cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment.Attractiveness←
DialogListener.connectToAP (
              MTrackerAP ap )
```

6.2.1.2 onUpdateAP()

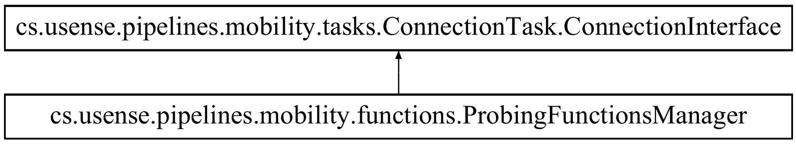
```
void cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment.Attractiveness←↩
DialogListener.onUpdateAP (
              MTrackerAP ap )
```

The documentation for this interface was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/fragments/ **AttractivenessDialogFragment.java**

6.3 cs.usense.pipelines.mobility.tasks.ConnectionTask.ConnectionInterface Interface Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.ConnectionTask.ConnectionInterface:



Public Member Functions

- void **connection** (int connection)

6.3.1 Detailed Description

Interface of this class used to communicate some results.

6.3.2 Member Function Documentation

6.3.2.1 connection()

```
void cs.usense.pipelines.mobility.tasks.ConnectionTask.ConnectionInterface.connection (
             int connection )
```

Method used to notify the status of the internet connectivity. 1 if ther is internet connection, otherwise 0.

Parameters

connection	

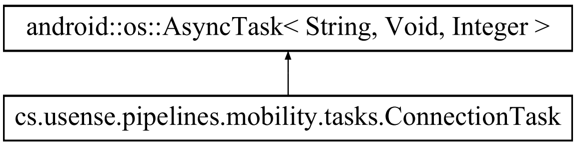
Implemented in **cs.usense.pipelines.mobility.functions.ProbingFunctionsManager** (p. 78).

The documentation for this interface was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **ConnectionTask.java**

6.4 cs.usense.pipelines.mobility.tasks.ConnectionTask Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.ConnectionTask:



Classes

- interface **ConnectionInterface**

Public Member Functions

- **ConnectionTask** (**ConnectionInterface** connectionInterface)

Protected Member Functions

- Integer **doInBackground** (String... sUrl)
- void **onPostExecute** (Integer result)

6.4.1 Detailed Description

This class extend to AsyncTask class and is used to verify if there is internet connectivity in the access point. To do this, a HTTP connection is establish to an specific URL. Created by copelabs on 08/09/2017.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ConnectionTask()

```
cs.usense.pipelines.mobility.tasks.ConnectionTask.ConnectionTask (
            ConnectionInterface connectionInterface )
```

InternetConnectionTAsk Constructor.

Parameters

connectionInterface	

6.4.3 Member Function Documentation

6.4.3.1 doInBackground()

```
Integer cs.usense.pipelines.mobility.tasks.ConnectionTask.doInBackground (
            String... sUrl ) [protected]
```

This method is executed in background and its function is to establish a HTTP connection to a specific server.

Parameters

sUrl	URL to connect.

Returns

6.4.3.2 onPostExecute()

```
void cs.usense.pipelines.mobility.tasks.ConnectionTask.onPostExecute (
            Integer result ) [protected]
```

When the task ends the interface is notify with the status of the internet connectivity.

Parameters

result	1 if there is internet connection, otherwise 0.

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **ConnectionTask.java**

6.5 cs.usense.pipelines.mobility.interfaces.DataBaseChangeListener Interface Reference

Public Member Functions

- void **onDataBaseChange** (List< **MTrackerAP** > apEntries)
- void **onStatusMessageChange** (String newMessage)

6.5.1 Member Function Documentation

6.5.1.1 onDataBaseChange()

```
void cs.usense.pipelines.mobility.interfaces.DataBaseChangeListener.onDataBaseChange (
            List< MTrackerAP > apEntries )
```

6.5.1.2 onStatusMessageChange()

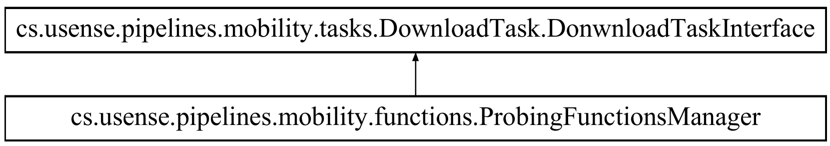
```
void cs.usense.pipelines.mobility.interfaces.DataBaseChangeListener.onStatusMessageChange (
            String newMessage )
```

The documentation for this interface was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/interfaces/ **DataBaseChangeListener.java**

6.6 cs.usense.pipelines.mobility.tasks.DownloadTask.DonwnloadTaskInterface Interface Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.DownloadTask.DonwnloadTaskInterface:



Public Member Functions

- void **donwloadTime** (float networkUtilization)

6.6.1 Detailed Description

Interface used to notify actions from this class.

6.6.2 Member Function Documentation

6.6.2.1 donwloadTime()

```
void cs.usense.pipelines.mobility.tasks.DownloadTask.DonwnloadTaskInterface.donwloadTime (
            float networkUtilization )
```

This method is used to notify the network utilization calculated by this task.

Parameters

networkUtilization	

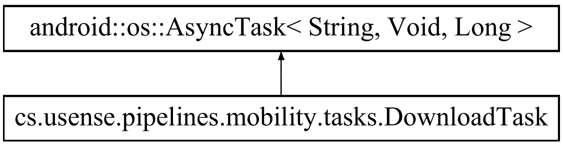
Implemented in **cs.usense.pipelines.mobility.functions.ProbingFunctionsManager** (p. 79).

The documentation for this interface was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **DownloadTask.java**

6.7 cs.usense.pipelines.mobility.tasks.DownloadTask Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.DownloadTask:



Classes

- interface **DonwnloadTaskInterface**

Public Member Functions

- **DownloadTask** (**DonwnloadTaskInterface** donwnloadTaskInterface, int attemps)

Protected Member Functions

- Long **doInBackground** (String... sUrl)
- void **onPostExecute** (Long result)

6.7.1 Detailed Description

Class is an asynctask class which is used to calculate network utilization. To do this, a file is downloaded from a server and is counting the time used to do it. Created by copelabs on 08/09/2017.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 DownloadTask()

```
cs.usense.pipelines.mobility.tasks.DownloadTask.DownloadTask (
              DonwnloadTaskInterface donwnloadTaskInterface,
              int attemps )
```

Downloadtask Constructor.

Parameters

donwnloadTaskInterface	interface of the task.
attemps	number of attempts.

6.7.3 Member Function Documentation

6.7.3.1 doInBackground()

```
Long cs.usense.pipelines.mobility.tasks.DownloadTask.doInBackground (
            String... sUrl ) [protected]
```

This method downloads a file from a server ann is counting the time used to do it.

Parameters

sUrl	

Returns

6.7.3.2 onPostExecute()

```
void cs.usense.pipelines.mobility.tasks.DownloadTask.onPostExecute (
            Long result ) [protected]
```

When the task end, this method is called and is calculated the network utilization of the AP.

Parameters

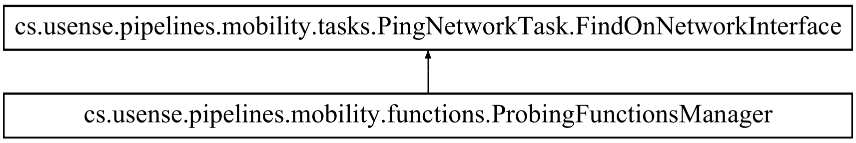
result	Time used to download the file.

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **DownloadTask.java**

6.8 cs.usense.pipelines.mobility.tasks.PingNetworkTask.FindOnNetworkInterface Interface Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.PingNetworkTask.FindOnNetworkInterface:



Public Member Functions

- void **networkFinder** (int devices)

6.8.1 Detailed Description

Interface used to notify action from this class.

6.8.2 Member Function Documentation

6.8.2.1 networkFinder()

```
void cs.usense.pipelines.mobility.tasks.PingNetworkTask.FindOnNetworkInterface.networkFinder (
            int devices )
```

This method is used to notify when the task ends.

Parameters

devices	Number of devices connected to the AP.

Implemented in **cs.usense.pipelines.mobility.functions.ProbingFunctionsManager** (p. 79).

The documentation for this interface was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **PingNetworkTask.java**

6.9 cs.usense.pipelines.mobility.functions.Functions Class Reference

Static Public Member Functions

- static double **function01** (long visits, int rejections, float timeLastConnectionAvrg, float timeAdvg, float attractiveness)
- static double **function0** (long visits, int rejections, float timeLastConnectionAvrg, float timeAdvg, float attractiveness)
- static double **function1** (int connectivity, float networkUtilization, long numDevices, int quality)
- static double **function2** (int connectivity, float networkUtilization, int numRecommendations, int quality)

- static double **function3** (int connectivity, float networkUtilization, int numRecommendations, double recommendations, int quality)
- static double **function4** (long visits, int rejections, float timeLastConnectionAvrg, float timeAdvg, float attractiveness, double recommendations)
- static float **functionGammaTimeConnection** (float timeAvrg, long actualTime, long time)
- static float **functionGammaTimeDisconnection** (float timeAvrg, long actualTime)
- static double **functionGammaRank** (double rankAvg, double rank)
- static double **sumRank3** (Map< String, Double > mapRank)
- static int **countOccurences** (Map< String, String > table, String value)
- static double **sumRank4** (Map< String, Double > mapRank, double rankIJ)

6.9.1 Detailed Description

This class provides a set o fucntion that can be used to calculate the ranking value of an access point.

Author

Omar Aponte (ULHT)

Version

3.0

Created by copelabs on 16/10/2017.

6.9.2 Member Function Documentation

6.9.2.1 countOccurences()

```
static int cs.usense.pipelines.mobility.functions.Functions.countOccurences (
            Map< String, String > table,
            String value ) [static]
```

This function return the number of recommendation received, this values is used in function 2.

Parameters

table	Map with avery recommendation received.
value	Value of the SSID of the acces point actually active.

Returns

Sum of number of entry in the map.

6.9.2.2 function0()

```
static double cs.usense.pipelines.mobility.functions.Functions.function0 (
            long visits,
            int rejections,
            float timeLastConnectionAvrg,
            float timeAdvg,
            float attractiveness ) [static]
```

Function 0. Function without probing.

Parameters

visits	Number of visits.
rejections	Number of rejections.
timeLastConnectionAvrg	Gap time connection (EMA).
timeAdvg	Time connection (EMA).
attractiveness	Attractiveness of the AP.

Returns

Function result.

6.9.2.3 function01()

```
static double cs.usense.pipelines.mobility.functions.Functions.function01 (
            long visits,
            int rejections,
            float timeLastConnectionAvrg,
            float timeAdvg,
            float attractiveness ) [static]
```

6.9.2.4 function1()

```
static double cs.usense.pipelines.mobility.functions.Functions.function1 (
            int connectivity,
            float networkUtilization,
            long numDevices,
            int quality ) [static]
```

Function 1 with Probing.

Parameters

connectivity	Internet connection status.
networkUtilization	Network utilization.
numDevices	Number of devices active (with Ip) connected to the AP.
quality	Quality of the signal.

Returns

Result of function 1.

6.9.2.5 function2()

```
static double cs.usense.pipelines.mobility.functions.Functions.function2 (
            int connectivity,
            float networkUtilization,
            int numRecommendations,
            int quality )  [static]
```

Function 2 with probing and recommendations.

Parameters

connectivity	Internet connection status.
networkUtilization	Network utilization.
numRecommendations	Number of recommendation received from other devices.
quality	Quality of the signal.

Returns

Result of function 2.

6.9.2.6 function3()

```
static double cs.usense.pipelines.mobility.functions.Functions.function3 (
            int connectivity,
            float networkUtilization,
            int numRecommendations,
            double recommendations,
            int quality )  [static]
```

Function 3 with probing and recommendation.

Parameters

connectivity	Internet connection status.
networkUtilization	Network utilization.
numRecommendations	Numer of recommendations received from others devices.
recommendations	Sum of the recommendation values received from others devices.
quality	Quality of the signal.

Returns

Result of the function 3.

6.9.2.7 function4()

```
static double cs.usense.pipelines.mobility.functions.Functions.function4 (
            long visits,
            int rejections,
            float timeLastConnectionAvrg,
            float timeAdvg,
            float attractiveness,
            double recommendations )  [static]
```

Function 4 without probing and with recommendations.

Parameters

visits	Number of visits.
rejections	Number of rejections.
timeLastConnectionAvrg	Gap average connection (EMA).
timeAdvg	Time connection (EMA).
attractiveness	Attractiveness of the AP.
recommendations	Recommendation receive from others devices. Result of Centrality function.

Returns

Result of function 4.

6.9.2.8 functionGammaRank()

```
static double cs.usense.pipelines.mobility.functions.Functions.functionGammaRank (
            double rankAvg,
            double rank )  [static]
```

Function used to calculate the EMA value of the rank.

Parameters

rankAvg	Rank average previously calculated.
rank	Actual rank value

Returns

Result of EMA function.

6.9.2.9 functionGammaTimeConnection()

```
static float cs.usense.pipelines.mobility.functions.Functions.functionGammaTimeConnection (
            float timeAvrg,
            long actualTime,
            long time )  [static]
```

Function used to calculate the EMA function using the time connected to a specific access point.

Parameters

timeAvrg	Time average previously calculated.
actualTime	Actual time connection.
time	Actual time.

Returns

Result of EMA function.

6.9.2.10 functionGammaTimeDisconnection()

```
static float cs.usense.pipelines.mobility.functions.Functions.functionGammaTimeDisconnection (
            float timeAvrg,
            long actualTime )  [static]
```

Function used to calculate the EMA function of the disconnected time.(Gap time between connections).

Parameters

timeAvrg	Time average previously calculated.
actualTime	Actual time connected.

Returns

Result of EMA function.

6.9.2.11 sumRank3()

```
static double cs.usense.pipelines.mobility.functions.Functions.sumRank3 (
            Map< String, Double > mapRank )  [static]
```

This fucntion is used to sum the rank values received from others devices.

Parameters

mapRank	Map whit the values received.

Returns

Sum of every value.

6.9.2.12 sumRank4()

```
static double cs.usense.pipelines.mobility.functions.Functions.sumRank4 (
            Map< String, Double > mapRank,
            double rankIJ ) [static]
```

This function calculates the sum of all ranking values received fron other device related to function 4.

Parameters

mapRank	Map with every rank received from others devices.
rankIJ	Actual value rank of the access point actual connected.

Returns

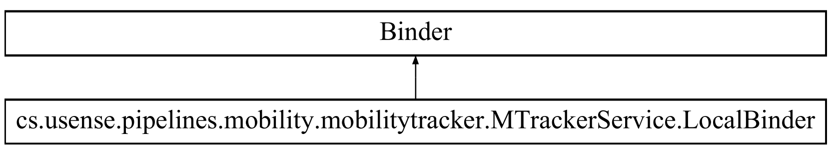
retur the sum of the rank values. Using centrality function.

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/functions/ **Functions.java**

6.10 cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.LocalBinder Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.LocalBinder:



Public Member Functions

- **MTrackerService getService** ()

6.10.1 Member Function Documentation

6.10.1.1 getService()

```
MTrackerService cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.LocalBinder.↩
getService ( )
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/mobilitytracker/ **MTrackerService.java**

6.11 cs.usense.pipelines.mobility.models.MTrackerAP Class Reference

Public Member Functions

- **MTrackerAP** ()
- int **getRejected** ()
- void **setRejected** (int rejected)
- double **getRank** ()
- void **setRank** (double rank)
- int **getRejections** ()
- void **setRejections** (int rejections)
- float **getNetworkUtilization** ()
- void **setNetworkUtilization** (float networkUtilization)
- int **getDevicesOnNetwork** ()
- void **setDevicesOnNetwork** (int devices)
- int **getConnection** ()
- void **setConnection** (int connect)
- int **getQuality** ()
- void **setQuality** (int quality)
- double **getRecommendation** ()
- void **setRecommendation** (double recommendation)
- int **getNumRecommendations** ()
- void **setNumRecommendations** (int numRecommendations)
- String **getSSID** ()
- void **setSSID** (String sSID)
- String **getBSSID** ()
- void **setBSSID** (String bSSID)
- double **getAttractiveness** ()
- void **setAttractiveness** (double attractiveness)
- String **getLastGatewayIp** ()
- void **setLastGatewayIp** (String lastGatewayIp)
- void **setLastGatewayIp** (int lastGatewayIp)
- void **setToDefault** (double uloopDispositionalTrust)
- String **toString** ()

6.11.1 Detailed Description

This class represents what MTracker considers an AP. The information kept in this object are mSsid, mBssid, m↩
Attractiveness, and last IP.

Author

Jonnahtan Saltarin (ULHT)
Rute Sofia (ULHT)
Christian da Silva Pereira (ULHT)
Luis Amaral Lopes (ULHT)
Omar Aponte (ULHT)

Version

3.0

6.11.2 Constructor & Destructor Documentation

6.11.2.1 MTrackerAP()

```
cs.usense.pipelines.mobility.models.MTrackerAP.MTrackerAP ( )
```

MTracker AP Constructor

6.11.3 Member Function Documentation

6.11.3.1 getAttractiveness()

```
double cs.usense.pipelines.mobility.models.MTrackerAP.getAttractiveness ( )
```

Get the mAttractiveness of this AP

Returns

the mAttractiveness

6.11.3.2 getBSSID()

```
String cs.usense.pipelines.mobility.models.MTrackerAP.getBSSID ( )
```

Get the mBssid of this AP

Returns

the bSSID

6.11.3.3 getConnection()

```
int cs.usense.pipelines.mobility.models.MTrackerAP.getConnection ( )
```

Get mConnection condition of this AP

Returns

Value 1 if there is internet mConnection.

6.11.3.4 getDevicesOnNetwork()

```
int cs.usense.pipelines.mobility.models.MTrackerAP.getDevicesOnNetwork ( )
```

Get devices connected to this AP.

Returns

Number of devices connected.

6.11.3.5 getLastGatewayIp()

```
String cs.usense.pipelines.mobility.models.MTrackerAP.getLastGatewayIp ( )
```

Get the last IP shown by this AP

Returns

the mLastGatewayIp

6.11.3.6 getNetworkUtilization()

```
float cs.usense.pipelines.mobility.models.MTrackerAP.getNetworkUtilization ( )
```

Get the Network Utilization of this AP.

Returns

Network utilization value.

6.11.3.7 getNumRecommendations()

```
int cs.usense.pipelines.mobility.models.MTrackerAP.getNumRecommendations ( )
```

Get number of recommendations of this AP.

Returns

Number of recommendations.

6.11.3.8 getQuality()

```
int cs.usense.pipelines.mobility.models.MTrackerAP.getQuality ( )
```

Get signal mQuality of thif AP.

Returns

Quality of this AP.

6.11.3.9 getRank()

```
double cs.usense.pipelines.mobility.models.MTrackerAP.getRank ( )
```

Get the value mRank of this AP.

Returns

value mRank.

6.11.3.10 getRecommendation()

```
double cs.usense.pipelines.mobility.models.MTrackerAP.getRecommendation ( )
```

Get mRecommendation value of this AP.

Returns

Recommendation value.

6.11.3.11 getRejected()

```
int cs.usense.pipelines.mobility.models.MTrackerAP.getRejected ( )
```

Get information of if the AP was mRejected.

Returns

Value 1 if was mRejected.

6.11.3.12 getRejections()

```
int cs.usense.pipelines.mobility.models.MTrackerAP.getRejections ( )
```

Get the number of mRejections of this AP.

Returns

Number of rejection.

6.11.3.13 getSSID()

```
String cs.usense.pipelines.mobility.models.MTrackerAP.getSSID ( )
```

Get the mSsid of this AP

Returns

the sSID

6.11.3.14 setAttractiveness()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setAttractiveness (
            double attractiveness )
```

Set the mAttractiveness of this AP

Parameters

attractiveness	the mAttractiveness to set

6.11.3.15 setBSSID()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setBSSID (
            String bSSID )
```

Set the mBssid of this AP

Parameters

bSSID	the bSSID to set

6.11.3.16 setConnection()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setConnection (
            int connect )
```

Set mConnection condition of this AP.

Parameters

connect	Value 1 if there is internet mConnection.

6.11.3.17 setDevicesOnNetwork()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setDevicesOnNetwork (
            int devices )
```

Set devices connected to this AP.

Parameters

devices	Numbr of devices connected to this AP.

6.11.3.18 setLastGatewayIp() [1/2]

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setLastGatewayIp (
            String lastGatewayIp )
```

Set the last IP shown by this AP

Parameters

last↩ GatewayIp	String representing the last gateway IP to set

6.11.3.19 setLastGatewayIp() [2/2]

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setLastGatewayIp (
            int lastGatewayIp )
```

Set the last IP shown by this AP

Parameters

last↩ GatewayIp	Integer representing the last gateway IP to set

6.11.3.20 setNetworkUtilization()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setNetworkUtilization (
            float networkUtilization )
```

Set the Network Utilization of this AP.

Parameters

networkUtilization	Network utilization value.

6.11.3.21 setNumRecommendations()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setNumRecommendations (
            int numRecommendations )
```

Set number of recommendations of this AP.

Parameters

numRecommendations	number of mRecommendation.

6.11.3.22 setQuality()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setQuality (
            int quality )
```

Set Signal mQuality of this AP.

Parameters

quality	Quality value of this AP.

6.11.3.23 setRank()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setRank (
            double rank )
```

Set the value of the mRank of this AP.

Parameters

rank	ProbingFunctionsManager value.

6.11.3.24 setRecommendation()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setRecommendation (
            double recommendation )
```

Set mRecommendation to this AP.

Parameters

recommendation	recommendations value.

6.11.3.25 setRejected()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setRejected (
            int rejected )
```

Set if this AP was mRejected.

Parameters

rejected	Value 1 if AP was mRejected.

6.11.3.26 setRejections()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setRejections (
            int rejections )
```

Set the number of rejection of this AP.

Parameters

rejections	Number of mRejections.

6.11.3.27 setSSID()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setSSID (
            String sSID )
```

Set the mSsid of this AP

Parameters

sSID	the sSID to set

6.11.3.28 setToDefault()

```
void cs.usense.pipelines.mobility.models.MTrackerAP.setToDefault (
            double uloopDispositionalTrust )
```

Set some parameters to default

6.11.3.29 toString()

```
String cs.usense.pipelines.mobility.models.MTrackerAP.toString ( )
```

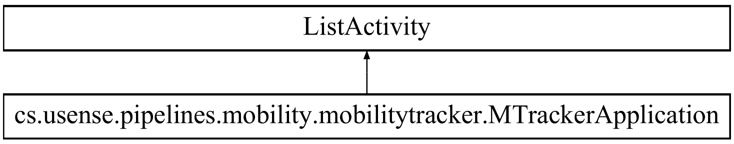
Return a string containing the mSsid, mBssid and the Attractiveness.

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/models/ **MTrackerAP.java**

6.12 cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication:



Public Member Functions

- boolean **onCreateOptionsMenu** (Menu menu)
- boolean **onOptionsItemSelected** (MenuItem item)

Protected Member Functions

- void **onCreate** (Bundle savedInstanceState)
- void **onStart** ()
- void **onStop** ()
- void **onDestroy** ()

6.12.1 Detailed Description

The Class **MTrackerApplication** (p. 37) is the front-end of the android application. It extends ListActivity to show the list of visited access points as a ListView. It starts the MTracker server, if not running already.

Author

Jonnahtan Saltarin (ULHT)
Rute Sofia (ULHT)
Christian da Silva Pereira (ULHT)
Luis Amaral Lopes (ULHT)

Version

3.0

6.12.2 Member Function Documentation

6.12.2.1 onCreate()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication.onCreate (
            Bundle savedInstanceState )  [protected]
```

6.12.2.2 onCreateOptionsMenu()

```
boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication.onCreateOptionsMenu (
            Menu menu )
```

6.12.2.3 onDestroy()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication.onDestroy ( )  [protected]
```

6.12.2.4 onOptionsItemSelected()

```
boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication.onOptionsItemSelected
(
            MenuItem item )
```

6.12.2.5 onStart()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication.onStart ( )  [protected]
```

6.12.2.6 onStop()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication.onStop ( )  [protected]
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/mobilitytracker/ **MTrackerApplication.java**

6.13 cs.usense.pipelines.mobility.helpers.MTrackerDataSource Class Reference

Public Member Functions

- **MTrackerDataSource** (Context context)
- void **openDB** (boolean writable) throws SQLException
- void **closeDB** ()
- synchronized long **registerNewRank** (**MTrackerAP** ap, long connectionUptime, long gapConnection, float gamma, float gammaGap, int function, double rank, double gammaRank, String time, String date, float battery)
- long **getNumAP** ()
- long **registerNewAP** (**MTrackerAP** ap)
- boolean **updateAP** (**MTrackerAP** ap)
- boolean **updateAttractivenessAP** (**MTrackerAP** ap)
- boolean **updateAPRejected** (**MTrackerAP** ap)
- boolean **updateParameters** (**MTrackerAP** ap)
- **MTrackerAP** **getAP** (String bssid)
- boolean **hasAP** (String bssid)
- Map< String, **MTrackerAP** > **getAllAP** ()
- Map< String, **MTrackerAP** > **getAllAP** (List< ScanResult > availableAP)
- void **getAllRANK** ()
- **MTrackerAP** **getBestAP** ()
- **MTrackerAP** **getBestAP** (List< ScanResult > availableAP)
- void **writeAPListToFile** ()
- void **writeRankingListToFile** ()
- long **getStationaryTime** (**MTrackerAP** ap)
- long **getLastDesconnection** (**MTrackerAP** ap)
- long **getStationaryTimeByMoment** (**MTrackerAP** ap, int dayOfTheWeek)
- long **countVisits** (**MTrackerAP** ap)
- long **rejectConnections** (**MTrackerAP** ap)
- long **devicesOnNetwork** (**MTrackerAP** ap)
- double **getRank** (**MTrackerAP** ap)
- double **getRank** (**MTrackerAP** ap, int function)
- double **getRankEMA** (**MTrackerAP** ap, int function)
- double **getInstantaneousRank** (**MTrackerAP** ap, Long currentDuration)
- long **registerNewVisit** (String SSID, String BSSID, Long startTime, Long endTime)
- boolean **updateVisit** (long _id, String SSID, String BSSID, Long startTime, Long endTime)
- List< **MTrackerVisit** > **getAllVisits** ()
- List< String > **getAllVisitsString** (**MTrackerAP** ap)
- long **getLastVisitDuration** (**MTrackerAP** ap)
- long **getLastMesurement** (**MTrackerAP** ap)
- float **getLastGAMMA** (**MTrackerAP** ap)
- float **getLastGAMMA** (**MTrackerAP** ap, int function)
- float **getLastGAMMAGAP** (**MTrackerAP** ap, int function)
- float **getLastGammaRank** (**MTrackerAP** ap, int function)
- long **getNumVisits** ()
- void **writeVisitListToFile** ()

6.13.1 Detailed Description

This class provides methods to insert, update and query the application database. It also provide methods to compute certain values, like the ProbingFunctionsManager and the Stationary Time, among others.

Author

Jonnahtan Saltarin (ULHT)
Rute Sofia (ULHT)
Christian da Silva Pereira (ULHT)
Luis Amaral Lopes (ULHT)

Version

3.0

6.13.2 Constructor & Destructor Documentation

6.13.2.1 MTrackerDataSource()

```
cs.usense.pipelines.mobility.helpers.MTrackerDataSource.MTrackerDataSource (
            Context context )
```

Constructor that takes Android Context as input.

Parameters

context	

6.13.3 Member Function Documentation

6.13.3.1 closeDB()

```
void cs.usense.pipelines.mobility.helpers.MTrackerDataSource.closeDB ( )
```

Close the predefined MTracker database.

6.13.3.2 countVisits()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.countVisits (
            MTrackerAP ap )
```

Computes the Number of visits that the node has done to a given AP.

Parameters

ap	The MTrackerAP whose Stationary Time is to be computed.

Returns

The number of visits.

6.13.3.3 devicesOnNetwork()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.devicesOnNetwork (
        MTrackerAP ap )
```

6.13.3.4 getAllAP() [1/2]

```
Map<String, MTrackerAP> cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getAllAP ( )
```

Gets the all the AP recorded by the application on the ACCESS_POINTS table.

Returns

A map with the AP objects, and the bssid as key.

6.13.3.5 getAllAP() [2/2]

```
Map<String, MTrackerAP> cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getAllAP (
        List< ScanResult > availableAP )
```

Gets the all the AP recorded by the application on the ACCESS_POINTS table, and the return only the ones that are also available in the List of ScanResult.

Returns

A map with the AP objects, and the bssid as key.

6.13.3.6 getAllRANK()

```
void cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getAllRANK ( )
```

Gets the all ranking recorded by the application on the RANKING table, and the return only the ones that are also available in the List of ScanResult.

Returns

A map with the AP objects, and the bssid as key.

6.13.3.7 getAllVisits()

```
List< MTrackerVisit> cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getAllVisits ( )
```

Get a List with all the visit objects stored in the database.

6.13.3.8 getAllVisitsString()

```
List<String> cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getAllVisitsString (
             MTrackerAP ap )
```

6.13.3.9 getAP()

```
MTrackerAP cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getAP (
             String bssid )
```

Gets an AP already registered by the application.

Parameters

bssid	The ssid of the AP which information should be returned

Returns

the MTrackerAP object, null if not found.

6.13.3.10 getBestAP() [1/2]

```
MTrackerAP cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getBestAP ( )
```

Checks all the AP registered by the application and return the one with the highest ProbingFunctionsManager.

Returns

the best AP registered by the application.

6.13.3.11 getBestAP() [2/2]

```
MTrackerAP cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getBestAP (
        List< ScanResult > availableAP )
```

Checks the APs registered by the application and available in the List of ScanResult, and the return the one with the highest probingFunctionsManager.

Returns

the best AP registered by the application.

6.13.3.12 getInstantaneousRank()

```
double cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getInstantaneousRank (
        MTrackerAP ap,
        Long currentDuration )
```

Test Method to compute the ProbingFunctionsManager of this node towards a given AP, taking into consideration the current visit time.

Parameters

ap	The MTrackerAP whose Stationary Time is to be computed.
currentDuration	current connection time.

Returns

The number of visits.

6.13.3.13 getLastDesconnection()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getLastDesconnection (
        MTrackerAP ap )
```

6.13.3.14 getLastGAMMA() [1/2]

```
float cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getLastGAMMA (
              MTrackerAP ap )
```

6.13.3.15 getLastGAMMA() [2/2]

```
float cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getLastGAMMA (
              MTrackerAP ap,
             int function )
```

6.13.3.16 getLastGAMMAGAP()

```
float cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getLastGAMMAGAP (
              MTrackerAP ap,
             int function )
```

6.13.3.17 getLastGammaRank()

```
float cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getLastGammaRank (
              MTrackerAP ap,
             int function )
```

6.13.3.18 getLastMesurement()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getLastMesurement (
              MTrackerAP ap )
```

6.13.3.19 getLastVisitDuration()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getLastVisitDuration (
              MTrackerAP ap )
```

6.13.3.20 getNumAP()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getNumAP ( )
```

Gets the number of records in the ACCESS_POINTS table. This is, the number of AP registered on the application.

Returns

the number of AP registered by the application.

6.13.3.21 getNumVisits()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getNumVisits ( )
```

Get the number of visits registered in the database.

6.13.3.22 getRank() [1/2]

```
double cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getRank (
              MTrackerAP ap )
```

Computes the ProbingFunctionsManager of this node towards a given AP. The ProbingFunctionsManager is computed as

Parameters

ap	The MTrackerAP whose Stationary Time is to be computed.

Returns

The number of visits.

6.13.3.23 getRank() [2/2]

```
double cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getRank (
              MTrackerAP ap,
             int function )
```

6.13.3.24 getRankEMA()

```
double cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getRankEMA (
              MTrackerAP ap,
             int function )
```

6.13.3.25 getStationaryTime()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getStationaryTime (
            MTrackerAP ap )
```

Computes the Stationary Time for a given AP.

Parameters

ap	The MTrackerAP whose Stationary Time is to be computed.

Returns

The stationary time for the given AP.

6.13.3.26 getStationaryTimeByMoment()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.getStationaryTimeByMoment (
            MTrackerAP ap,
            int dayOfTheWeek )
```

Computes the Stationary Time for a given AP, only taking into consideration records for a given Day of the Week.

Parameters

ap	The MTrackerAP whose Stationary Time is to be computed.
dayOfTheWeek	Day of the week that will restrict the computation of the stationary time.

Returns

The stationary time for the given AP.

6.13.3.27 hasAP()

```
boolean cs.usense.pipelines.mobility.helpers.MTrackerDataSource.hasAP (
            String bssid )
```

Checks if a given AP has already been registered by the application.

Parameters

bssid	The ssid of the AP

Returns

true, if AP has already been registered by the application, false otherwise.

6.13.3.28 openDB()

```
void cs.usense.pipelines.mobility.helpers.MTrackerDataSource.openDB (
            boolean writable ) throws SQLException
```

Opens the predefined MTracker database.

Parameters

writable	

Exceptions

SQLException	

6.13.3.29 registerNewAP()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.registerNewAP (
            MTrackerAP ap )
```

Register a new AP in the application. It creates a new record on the ACCESS_POINTS table, with the information passed as MTrackerAP.

Parameters

ap	Access point information.

Returns

the row ID of the newly inserted row, or -1 if an error occurred.

6.13.3.30 registerNewRank()

```
synchronized long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.registerNewRank (
            MTrackerAP ap,
            long connectionUptime,
            long gapConnection,
            float gamma,
```

```
float gammaGap,
int function,
double rank,
double gammaRank,
String time,
String date,
float battery )
```

6.13.3.31 registerNewVisit()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.registerNewVisit (
            String SSID,
            String BSSID,
            Long startTime,
            Long endTime )
```

Register a new visit into the database.

Parameters

SSID	SSID
BSSID	BSSID
startTime	Time at which the connection started.
endTime	Time at which the connection ended.

Returns

id of the created record, -1 if an error occurs.

6.13.3.32 rejectConnections()

```
long cs.usense.pipelines.mobility.helpers.MTrackerDataSource.rejectConnections (
            MTrackerAP ap )
```

6.13.3.33 updateAP()

```
boolean cs.usense.pipelines.mobility.helpers.MTrackerDataSource.updateAP (
            MTrackerAP ap )
```

Update an AP already registered by the application. This modifies the corresponding record to the AP in the AC↩
CESS_POINTS table.

Parameters

ap	Access point information.

Returns

true, if successful.

6.13.3.34 updateAPRejected()

```
boolean cs.usense.pipelines.mobility.helpers.MTrackerDataSource.updateAPRejected (
              MTrackerAP ap )
```

6.13.3.35 updateAttractivenessAP()

```
boolean cs.usense.pipelines.mobility.helpers.MTrackerDataSource.updateAttractivenessAP (
              MTrackerAP ap )
```

6.13.3.36 updateParameters()

```
boolean cs.usense.pipelines.mobility.helpers.MTrackerDataSource.updateParameters (
              MTrackerAP ap )
```

6.13.3.37 updateVisit()

```
boolean cs.usense.pipelines.mobility.helpers.MTrackerDataSource.updateVisit (
             long _id,
             String SSID,
             String BSSID,
             Long startTime,
             Long endTime )
```

Updates an existing visit in the database.

Parameters

_id	id of the record to update
SSID	SSID
BSSID	BSSID
startTime	Time at which the connection started.
endTime	Time at which the connection ended.

Returns

id of the created record, -1 if an error occurs.

6.13.3.38 writeAPListToFile()

```
void cs.usense.pipelines.mobility.helpers.MTrackerDataSource.writeAPListToFile ( )
```

Write all the AP registered by the application into a text file (MTracker.txt), located in the root of the directory.

6.13.3.39 writeRankingListToFile()

```
void cs.usense.pipelines.mobility.helpers.MTrackerDataSource.writeRankingListToFile ( )
```

Write all the AP registered by the application into a text file (MTracker.txt), located in the root of the directory.

6.13.3.40 writeVisitListToFile()

```
void cs.usense.pipelines.mobility.helpers.MTrackerDataSource.writeVisitListToFile ( )
```

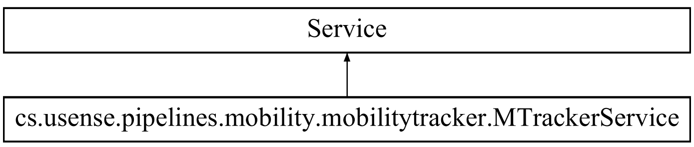
Writes the Visit List to the file MTrackerVisits.txt.

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/helpers/ **MTrackerDataSource.java**

6.14 cs.usense.pipelines.mobility.mobilitytracker.MTrackerService Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.mobilitytracker.MTrackerService:



Classes

- class **LocalBinder**
- class **MTrackerServiceWifiListener**

Public Member Functions

- List< **MTrackerAP** > **getData** ()
- void **startPeriodicScanning** ()
- void **stopPeriodicScanning** ()
- void **onCreate** ()
- int **onStartCommand** (Intent intent, int flags, int startId)
- void **onDestroy** ()
- IBinder **onBind** (Intent intent)
- void **setOnStateChangeListener** (**DataBaseChangeListener** listener)
- void **clearOnStateChangeListeners** ()
- void **notifyDataBaseChange** ()
- void **writeAPListToFile** ()
- void **writeVisitListToFile** ()
- void **writeRankingToFile** (String function)
- boolean **setUloopDispositionalTrust** (double uloopDT)

Public Attributes

- double **uloopDispositionalTrust** = 1.0
- **MTrackerWifiManager** **wifiManager**
- **MTrackerServiceWifiListener** **wifiListener**
- **MTrackerDataSource** **dataSource**

6.14.1 Detailed Description

This class is contains the core functionalities of the application. The **MTrackerService** (p. 50) will run in background, getting WI-FI parameters and storing the required information in the database.

Author

Jonnahtan Saltarin (ULHT)
Rute Sofia (ULHT)
Christian da Silva Pereira (ULHT)
Luis Amaral Lopes (ULHT)

Version

3.0

6.14.2 Member Function Documentation

6.14.2.1 clearOnStateChangeListeners()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.clearOnStateChangeListeners
( )
```

6.14.2.2 getData()

```
List< MTrackerAP > cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.getData ( )
```

6.14.2.3 notifyDataBaseChange()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.notifyDataBaseChange ( )
```

Notifies a database change to the listeners.

6.14.2.4 onBind()

```
IBinder cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.onBind (
            Intent intent )
```

6.14.2.5 onCreate()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.onCreate ( )
```

6.14.2.6 onDestroy()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.onDestroy ( )
```

6.14.2.7 onStartCommand()

```
int cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.onStartCommand (
            Intent intent,
            int flags,
            int startId )
```

6.14.2.8 setOnStateChangeListener()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.setOnStateChangeListener (
            DataBaseChangeListener listener )
```

6.14.2.9 setUloopDispositionalTrust()

```
boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.setUloopDispositional←
Trust (
            double uloopDT )
```

Sets the ULOOP Dispositional Trust, which is the default attractiveness.

Parameters

uloopDT	Uloop dispositional trust.

Returns

true if uloopDT is valid [0-1], false otherwise

6.14.2.10 startPeriodicScanning()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.startPeriodicScanning ( )
```

Starts the periodic scanning. This will call the adequate function in the **MTrackerWifiManager** (p. 70), which will start a scan periodically. The time between each scan is defined in the **MTrackerWifiManager** (p. 70) class.

6.14.2.11 stopPeriodicScanning()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.stopPeriodicScanning ( )
```

Stops the periodic scanning.

6.14.2.12 writeAPListToFile()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.writeAPListToFile ( )
```

Writes the AP list to a text file.

6.14.2.13 writeRankingToFile()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.writeRankingToFile (
            String function )
```

Write ranking to File

6.14.2.14 writeVisitListToFile()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.writeVisitListToFile ( )
```

Write visits to File

6.14.3 Member Data Documentation

6.14.3.1 dataSource

MTrackerDataSource cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.dataSource

6.14.3.2 uloopDispositionalTrust

double cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.uloopDispositionalTrust = 1.0

6.14.3.3 wifiListener

MTrackerServiceWifiListener cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.↩ wifiListener

6.14.3.4 wifiManager

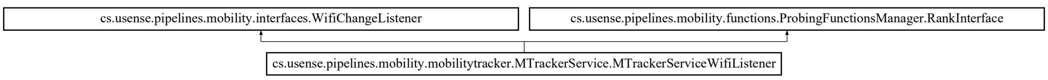
MTrackerWifiManager cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.wifiManager

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/mobilitytracker/ **MTrackerService.java**

6.15 cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerService↩ WifiListener Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifi↩ Listener:



Public Member Functions

- void **setMandatoryAP** (String SSID)
- **MTrackerServiceWifiListener** ()
- void **onWifiStateDisabled** (boolean valid, String bssid, String ssid, long visitId, long connectionStart, long connectionEnd)
- void **onWifiStateEnabled** ()
- void **onWifiConnectionDown** (boolean valid, String bssid, String ssid, long visitId, long connectionStart, long connectionEnd)
- long **onWifiConnectionUp** (String bssid, String ssid, List< ScanResult > lastScanResults)
- void **onWifiAvailableNetworksChange** (String bssid, List< ScanResult > results)
- void **onWifiAvailableList** (List< ScanResult > results)
- void **onConnectionRejected** (String bssid, String ssid)
- void **rank** (double rank, int function, **MTrackerAP** ap)

Public Attributes

- final boolean **COMPUTE_ACTIVE_FUNCTIONS** =false
- final boolean **COMPUTE_PASSIVE_FUNCTION_0** = false
- final boolean **COMPUTE_PASSIVE_FUNCTION_4** = false
- final boolean **COMPUTE_CALCULATE_BESTAP** = false
- final boolean **CONNECT_TO_BESTAP** =false
- int **calculations** =0
- int **statblishMandatoryConnection** =0
- **ProbingFunctionsManager** **probingFunctionsManager** = new **ProbingFunctionsManager**(this,txt↩ Record)
- String **mSsid**
- String **mAPMandatory**

6.15.1 Constructor & Destructor Documentation

6.15.1.1 MTrackerServiceWifiListener()

```
cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.M↩
TrackerServiceWifiListener ( )
```

6.15.2 Member Function Documentation

6.15.2.1 onConnectionRejected()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
onConnectionRejected (
            String bssid,
            String ssid )
```

Implements **cs.usense.pipelines.mobility.interfaces.WifiChangeListener** (p. 85).

6.15.2.2 onWifiAvailableList()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
onWifiAvailableList (
            List< ScanResult > results )
```

Implements **cs.usense.pipelines.mobility.interfaces.WifiChangeListener** (p. 85).

6.15.2.3 onWifiAvailableNetworksChange()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
onWifiAvailableNetworksChange (
            String bssid,
            List< ScanResult > results )
```

Implements **cs.usense.pipelines.mobility.interfaces.WifiChangeListener** (p. 85).

6.15.2.4 onWifiConnectionDown()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
onWifiConnectionDown (
            boolean valid,
            String bssid,
            String ssid,
            long visitId,
            long connectionStart,
            long connectionEnd )
```

Implements **cs.usense.pipelines.mobility.interfaces.WifiChangeListener** (p. 85).

6.15.2.5 onWifiConnectionUp()

```
long cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
onWifiConnectionUp (
            String bssid,
            String ssid,
            List< ScanResult > lastScanResults )
```

Implements **cs.usense.pipelines.mobility.interfaces.WifiChangeListener** (p. 85).

6.15.2.6 onWifiStateDisabled()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
onWifiStateDisabled (
            boolean valid,
            String bssid,
            String ssid,
            long visitId,
            long connectionStart,
            long connectionEnd )
```

Implements **cs.usense.pipelines.mobility.interfaces.WifiChangeListener** (p. 86).

6.15.2.7 onWifiStateEnabled()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
onWifiStateEnabled ( )
```

Implements **cs.usense.pipelines.mobility.interfaces.WifiChangeListener** (p. 86).

6.15.2.8 rank()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
rank (
            double rank,
            int function,
             MTrackerAP ap )
```

This funtion is used to notify when a rank function has a result.

Parameters

rank	Rank value.
function	Funtion used to calcule the rank.
ap	Access point information.

Implements **cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.RankInterface** (p. 81).

6.15.2.9 setMandatoryAP()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
setMandatoryAP (
            String SSID )
```

6.15.3 Member Data Documentation

6.15.3.1 calculations

```
int cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
calculations =0
```

6.15.3.2 COMPUTE_ACTIVE_FUNCTIONS

```
final boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerService↩
WifiListener.COMPUTE_ACTIVE_FUNCTIONS =false
```

6.15.3.3 COMPUTE_CALCULATE_BESTAP

```
final boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerService↩
WifiListener.COMPUTE_CALCULATE_BESTAP = false
```

6.15.3.4 COMPUTE_PASSIVE_FUNCTION_0

```
final boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerService↩
WifiListener.COMPUTE_PASSIVE_FUNCTION_0 = false
```

6.15.3.5 COMPUTE_PASSIVE_FUNCTION_4

```
final boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerService↩
WifiListener.COMPUTE_PASSIVE_FUNCTION_4 = false
```

6.15.3.6 CONNECT_TO_BESTAP

```
final boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerService↩
WifiListener.CONNECT_TO_BESTAP =false
```

6.15.3.7 mAPMandatory

```
String cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifi↩
Listener.mAPMandatory
```

6.15.3.8 mSsid

```
String cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifi↩
Listener.mSsid
```

6.15.3.9 probingFunctionsManager

```
ProbingFunctionsManager cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.M↩
TrackerServiceWifiListener.probingFunctionsManager = new ProbingFunctionsManager(this,txt↩
Record)
```

6.15.3.10 statblishMandatoryConnection

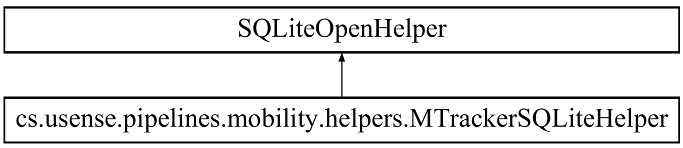
```
int cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener.↩
statblishMandatoryConnection =0
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/mobilitytracker/ **MTrackerService.java**

6.16 cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper:



Public Member Functions

- **MTrackerSQLiteHelper** (Context context)
- void **onCreate** (SQLiteDatabase dataBase)
- void **onUpgrade** (SQLiteDatabase dataBase, int oldVersion, int newVersion)

Static Public Attributes

- static final String **TABLE_ACCESSPOINTS** = "accesspoints"
- static final String **TABLE_VISITS** = "visits"
- static final String **TABLE_CONTEXT** = "context"
- static final String **TABLE_RANKING** = "ranking"
- static final String **COLUMN_ID** = "_id"
- static final String **COLUMN_SSID** = "ssid"
- static final String **COLUMN_BSSID** = "bssid"
- static final String **COLUMN_GROUPID** = "groupid"
- static final String **COLUMN_ATTRACTIVENESS** = "attractiveness"
- static final String **COLUMN_LASTGATEWAYIP** = "lastgatewayip"
- static final String **COLUMN_TIMEDOWNLOAD** = "timedownload"
- static final String **COLUMN_DEVICESONNETWORK** = "devicesonnetwork"
- static final String **COLUMN_REJECTIONS** = "rejections"

- static final String **COLUMN_REJECTED** = "rejected"
- static final String **COLUMN_TIMEON** = "timeon"
- static final String **COLUMN_TIMEOUT** = "timeout"
- static final String **COLUMN_DAYOFTHEWEEK** = "dayoftheweek"
- static final String **COLUMN_HOUR** = "hour"
- static final String **COLUMN_GANMA** = "ganma"
- static final String **COLUMN_RANK** = "probingFunctionsManager"
- static final String **COLUMN_VISITS** = "visits"
- static final String **COLUMN_VISIT_DURATION** = "visitduration"
- static final String **COLUMN_VISIT_GAP** = "visitgap"
- static final String **COLUMN_QUALITY** = "quality"
- static final String **COLUMN_CONNECTION** = "connection"
- static final String **COLUMN_RECOMMENCDATION** = "recommendation"
- static final String **COLUMN_NUM_RECOMMENDATIONS** = "numrecommendations"
- static final String **COLUMN_FUNCTION** = "function"
- static final String **COLUMN_GAMMA_GAP** = "gammagap"
- static final String **COLUMN_GAMMA_RANk** = "gammarank"
- static final String **COLUMN_TIME** = "time"
- static final String **COLUMN_DATE** = "date"
- static final String **COLUMN_BATTERY** = "battery"

6.16.1 Detailed Description

This class extends the SQLiteOpenHelper android class.

Author

Jonnahtan Saltarin (ULHT)
Rute Sofia (ULHT)
Christian da Silva Pereira (ULHT)
Luis Amaral Lopes (ULHT)

Version

3.0

6.16.2 Constructor & Destructor Documentation

6.16.2.1 MTrackerSQLiteHelper()

```
cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.MTrackerSQLiteHelper (
            Context context )
```

6.16.3 Member Function Documentation

6.16.3.1 onCreate()

```
void cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.onCreate (
            SQLiteDatabase dataBase )
```

6.16.3.2 onUpgrade()

```
void cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.onUpgrade (
            SQLiteDatabase dataBase,
            int oldVersion,
            int newVersion )
```

6.16.4 Member Data Documentation

6.16.4.1 COLUMN_ATTRACTIVENESS

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_ATTRACTIVENESS =
"attractiveness" [static]
```

6.16.4.2 COLUMN_BATTERY

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_BATTERY = "battery"
[static]
```

6.16.4.3 COLUMN_BSSID

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_BSSID = "bssid"
[static]
```

6.16.4.4 COLUMN_CONNECTION

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_CONNECTION =
"connection" [static]
```

6.16.4.5 COLUMN_DATE

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_DATE = "date" [static]

6.16.4.6 COLUMN_DAYOFTHEWEEK

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_DAYOFTHEWEEK = "dayoftheweek" [static]

6.16.4.7 COLUMN_DEVICESONNETWORK

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_DEVICESONNETWORK = "devicesonnetwork" [static]

6.16.4.8 COLUMN_FUNCTION

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_FUNCTION = "function" [static]

6.16.4.9 COLUMN_GAMMA_GAP

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_GAMMA_GAP = "gammagap" [static]

6.16.4.10 COLUMN_GAMMA_RANk

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_GAMMA_RANk = "gammarank" [static]

6.16.4.11 COLUMN_GANMA

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_GANMA = "ganma" [static]

6.16.4.12 COLUMN_GROUPID

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_GROUPID = "groupid" [static]

6.16.4.13 COLUMN_HOUR

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_HOUR = "hour" [static]

6.16.4.14 COLUMN_ID

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_ID = "_id" [static]

6.16.4.15 COLUMN_LASTGATEWAYIP

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_LASTGATEWAYIP = "lastgatewayip" [static]

6.16.4.16 COLUMN_NUM_RECOMMENDATIONS

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_NUM_RECOMMENDA↩TIONS = "numrecommendations" [static]

6.16.4.17 COLUMN_QUALITY

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_QUALITY = "quality" [static]

6.16.4.18 COLUMN_RANK

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_RANK = "probing↩FunctionsManager" [static]

6.16.4.19 COLUMN_RECOMMENCDATION

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_RECOMMENCDATION
= "recommendation"  [static]
```

6.16.4.20 COLUMN_REJECTED

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_REJECTED = "rejected"
[static]
```

6.16.4.21 COLUMN_REJECTIONS

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_REJECTIONS =
"rejections"  [static]
```

6.16.4.22 COLUMN_SSID

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_SSID = "ssid"
[static]
```

6.16.4.23 COLUMN_TIME

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_TIME = "time"
[static]
```

6.16.4.24 COLUMN_TIMEDOWNLOAD

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_TIMEDOWNLOAD =
"timedownload"  [static]
```

6.16.4.25 COLUMN_TIMEON

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_TIMEON = "timeon"
[static]
```

6.16.4.26 COLUMN_TIMEOUT

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_TIMEOUT = "timeout" [static]

6.16.4.27 COLUMN_VISIT_DURATION

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_VISIT_DURATION = "visitduration" [static]

6.16.4.28 COLUMN_VISIT_GAP

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_VISIT_GAP = "visitgap" [static]

6.16.4.29 COLUMN_VISITS

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.COLUMN_VISITS = "visits" [static]

6.16.4.30 TABLE_ACCESSPOINTS

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.TABLE_ACCESSPOINTS = "accesspoints" [static]

6.16.4.31 TABLE_CONTEXT

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.TABLE_CONTEXT = "context" [static]

6.16.4.32 TABLE_RANKING

final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.TABLE_RANKING = "ranking" [static]

6.16.4.33 TABLE_VISITS

```
final String cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper.TABLE_VISITS = "visits"
[static]
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/helpers/ **MTrackerSQLiteHelper.java**

6.17 cs.usense.pipelines.mobility.models.MTrackerVisit Class Reference

Public Member Functions

- String **getSSID** ()
- String **getBSSID** ()
- long **getStartTime** ()
- long **getEndTime** ()
- int **getDayOfTheWeek** ()
- int **getHourOfTheDay** ()
- void **setSSID** (String sSID)
- void **setBSSID** (String bSSID)
- void **setStartTime** (long startTime)
- void **setEndTime** (long endTime)
- void **setDayOfTheWeek** (int dayOfTheWeek)
- void **setHourOfTheDay** (int hourOfTheDay)
- **MTrackerVisit** ()
- void **setToDefault** ()
- void **update** (String SSID, String BSSID, Long startTime, Long endTime)
- String **toString** ()
- String **toStringTabFormat** ()

6.17.1 Detailed Description

This class represents what MTracker considers a Visit. The information kept in this object are SSID, BSSID, start and end time of the connection, the day of the week, and the hour of the day.

Author

Jonnahtan Saltarin (ULHT)
Rute Sofia (ULHT)
Christian da Silva Pereira (ULHT)
Luis Amaral Lopes (ULHT)
Omar Aponte (ULHT)

Version

3.0

6.17.2 Constructor & Destructor Documentation

6.17.2.1 MTrackerVisit()

```
cs.usense.pipelines.mobility.models.MTrackerVisit.MTrackerVisit ( )
```

6.17.3 Member Function Documentation

6.17.3.1 getBSSID()

```
String cs.usense.pipelines.mobility.models.MTrackerVisit.getBSSID ( )
```

Returns

the bSSID

6.17.3.2 getDayOfTheWeek()

```
int cs.usense.pipelines.mobility.models.MTrackerVisit.getDayOfTheWeek ( )
```

Returns

the dayOfTheWeek

6.17.3.3 getEndTime()

```
long cs.usense.pipelines.mobility.models.MTrackerVisit.getEndTime ( )
```

Returns

the endTime

6.17.3.4 getHourOfTheDay()

```
int cs.usense.pipelines.mobility.models.MTrackerVisit.getHourOfTheDay ( )
```

Returns

the hourOfTheDay

6.17.3.5 getSSID()

```
String cs.usense.pipelines.mobility.models.MTrackerVisit.getSSID ( )
```

Returns

the sSID

6.17.3.6 getStartTime()

```
long cs.usense.pipelines.mobility.models.MTrackerVisit.getStartTime ( )
```

Returns

the startTime

6.17.3.7 setBSSID()

```
void cs.usense.pipelines.mobility.models.MTrackerVisit.setBSSID (
            String bSSID )
```

Parameters

bSSID	the bSSID to set

6.17.3.8 setDayOfTheWeek()

```
void cs.usense.pipelines.mobility.models.MTrackerVisit.setDayOfTheWeek (
            int dayOfTheWeek )
```

Parameters

dayOfTheWeek	the dayOfTheWeek to set

6.17.3.9 setEndTime()

```
void cs.usense.pipelines.mobility.models.MTrackerVisit.setEndTime (
            long endTime )
```

Parameters

endTime	the endTime to set

6.17.3.10 setHourOfTheDay()

```
void cs.usense.pipelines.mobility.models.MTrackerVisit.setHourOfTheDay (
            int hourOfTheDay )
```

Parameters

hourOfTheDay	the hourOfTheDay to set

6.17.3.11 setSSID()

```
void cs.usense.pipelines.mobility.models.MTrackerVisit.setSSID (
            String sSID )
```

Parameters

sSID	the sSID to set

6.17.3.12 setStartTime()

```
void cs.usense.pipelines.mobility.models.MTrackerVisit.setStartTime (
            long startTime )
```

Parameters

startTime	the startTime to set

6.17.3.13 setToDefault()

```
void cs.usense.pipelines.mobility.models.MTrackerVisit.setToDefault ( )
```

6.17.3.14 toString()

```
String cs.usense.pipelines.mobility.models.MTrackerVisit.toString ( )
```

6.17.3.15 toStringTabFormat()

```
String cs.usense.pipelines.mobility.models.MTrackerVisit.toStringTabFormat ( )
```

6.17.3.16 update()

```
void cs.usense.pipelines.mobility.models.MTrackerVisit.update (
            String SSID,
            String BSSID,
            Long startTime,
            Long endTime )
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/models/ **MTrackerVisit.java**

6.18 cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager Class Reference

Classes

- class **WifiAvailableNetworksChange**
- class **WifiConnectionChange**
- class **WifiStateChange**

Public Member Functions

- void **connectToAP** (String networkSSID)
- void **connectToNewAP** (String ssid)
- void **close** (Context c)
- void **startPeriodicScanning** ()
- void **stopPeriodicScanning** ()
- void **setOnWifiChangeListener** (**WifiChangeListener** listener)
- void **clearOnWifiChangeListener** ()
- void **setWifiManager** (WifiManager wm)
- void **setWifiManager** (Context c)
- boolean **isWifiEnabled** ()
- boolean **startScan** ()
- List< ScanResult > **getLastScanResults** ()
- int **getGatewayIp** ()
- void **noteOngoingConnection** ()
- int **connectionQuality** ()
- String **ipCalculation** ()

Public Attributes

- boolean **isScanningActive** = false
- boolean **isWaitingScanResults** = false

Static Public Attributes

- static long **MINIMUM_CONNEXION_TIME** = 10
- static int **SCANNING_INTERVAL** = 10000

Protected Attributes

- long **wifiCurrentAPStart**

6.18.1 Detailed Description

This class provides some methods to provide extended functionality to the android WifiManager.

Author

Jonnahtan Saltarin (ULHT)
Rute Sofia (ULHT)
Christian da Silva Pereira (ULHT)
Luis Amaral Lopes (ULHT)

Version

3.0

6.18.2 Member Function Documentation

6.18.2.1 clearOnWifiChangeListener()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.clearOnWifiChange↩
Listener ( )
```

6.18.2.2 close()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.close (
             Context c )
```

6.18.2.3 connectionQuality()

```
int cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.connectionQuality ( )
```

6.18.2.4 connectToAP()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.connectToAP (
             String networkSSID )
```

6.18.2.5 connectToNewAP()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.connectToNewAP (
             String ssid )
```

6.18.2.6 getGatewayIp()

```
int cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.getGatewayIp ( )
```

6.18.2.7 getLastScanResults()

List<ScanResult> cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.getLast←
ScanResults ( )

6.18.2.8 ipCalculation()

String cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.ipCalculation ( )

6.18.2.9 isWifiEnabled()

boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.isWifiEnabled ( )

6.18.2.10 noteOngoingConnection()

void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.noteOngoingConnection (
)

6.18.2.11 setOnWifiChangeListener()

void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.setOnWifiChangeListener
(
WifiChangeListener *listener*)

6.18.2.12 setWifiManager() [1/2]

void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.setWifiManager (
WifiManager *wm*)

6.18.2.13 setWifiManager() [2/2]

void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.setWifiManager (
Context *c*)

6.18.2.14 startPeriodicScanning()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.startPeriodicScanning (
)
```

6.18.2.15 startScan()

```
boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.startScan ( )
```

6.18.2.16 stopPeriodicScanning()

```
void cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.stopPeriodicScanning ( )
```

6.18.3 Member Data Documentation

6.18.3.1 isScanningActive

```
boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.isScanningActive =
false
```

6.18.3.2 isWaitingScanResults

```
boolean cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.isWaitingScanResults
= false
```

6.18.3.3 MINIMUM_CONNEXION_TIME

```
long cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.MINIMUM_CONNEXION_TIME =
10 [static]
```

6.18.3.4 SCANNING_INTERVAL

```
int cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.SCANNING_INTERVAL = 10000
[static]
```

6.18.3.5 wifiCurrentAPStart

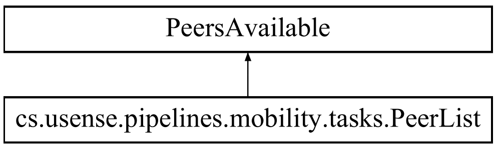
```
long cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.wifiCurrentAPStart [protected]
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/mobilitytracker/ **MTrackerWifiManager.java**

6.19 cs.usense.pipelines.mobility.tasks.PeerList Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.PeerList:



Public Member Functions

- **PeerList** ()
- void **onPeersAvailable** (WifiP2pDeviceList peers)

6.19.1 Constructor & Destructor Documentation

6.19.1.1 PeerList()

```
cs.usense.pipelines.mobility.tasks.PeerList.PeerList ( )
```

6.19.2 Member Function Documentation

6.19.2.1 onPeersAvailable()

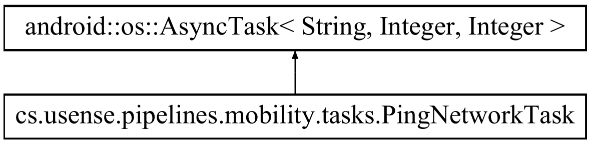
```
void cs.usense.pipelines.mobility.tasks.PeerList.onPeersAvailable (
            WifiP2pDeviceList peers )
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **PeerList.java**

6.20 cs.usense.pipelines.mobility.tasks.PingNetworkTask Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.PingNetworkTask:



Classes

- interface **FindOnNetworkInterface**

Public Member Functions

- **PingNetworkTask** (**FindOnNetworkInterface** findOnNetworkInterface)

Protected Member Functions

- Integer **doInBackground** (String... host)
- void **onPostExecute** (Integer result)

6.20.1 Detailed Description

This class is an async tack class used to executed ping into the network.

Created by copelabs on 27/04/2017.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 PingNetworkTask()

```
cs.usense.pipelines.mobility.tasks.PingNetworkTask.PingNetworkTask (
             FindOnNetworkInterface findOnNetworkInterface )
```

PingNetworkTask (p. 76) constructor.

Parameters

findOnNetworkInterface	Interface of the task.

6.20.3 Member Function Documentation

6.20.3.1 doInBackground()

```
Integer cs.usense.pipelines.mobility.tasks.PingNetworkTask.doInBackground (
            String... host ) [protected]
```

This method executes in background the task in the network.

Parameters

host	Host of the access point.

Returns

6.20.3.2 onPostExecute()

```
void cs.usense.pipelines.mobility.tasks.PingNetworkTask.onPostExecute (
            Integer result ) [protected]
```

When the task ends a interface is notify with the number os the devices connected to the access point.

Parameters

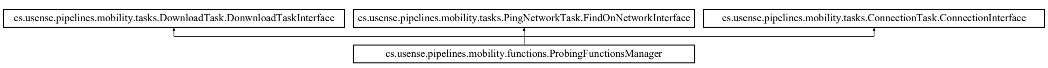
result	Number of devices connected.

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **PingNetworkTask.java**

6.21 cs.usense.pipelines.mobility.functions.ProbingFunctionsManager Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.functions.ProbingFunctionsManager:



Classes

- interface **RankInterface**

Public Member Functions

- **ProbingFunctionsManager** (**RankInterface** rankInterface, **TxtRecord** txtRecord)
- void **startRankingCalulation** (String accessPointIp, **MTrackerAP** ap)
- void **connection** (int connection)
- void **donwloadTime** (float networkUtilization)
- void **networkFinder** (int devices)
- boolean **isComputing** ()
- void **setIsComputing** (boolean computing)

Public Attributes

- final boolean **COMPUTE_FUNCTION_1** = false
- final boolean **COMPUTE_FUNCTION_2** = false
- final boolean **COMPUTE_FUNCTION_3** = false

6.21.1 Detailed Description

This function contains the functionalities to compute the fucntion with probing. From here are computed the task to calculated network utilization and number of active devices connected to the AP. Created by copelabs on 10/10/2017.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 ProbingFunctionsManager()

```
cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.ProbingFunctionsManager (
            RankInterface rankInterface,
            TxtRecord txtRecord )
```

ProbingFunctionManager constructor.

Parameters

rankInterface	Rank interface.
txtRecord	Txt record object.

6.21.3 Member Function Documentation

6.21.3.1 connection()

```
void cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.connection (
            int connection )
```

Method used to notify the status of the internet connectivity. 1 if ther is internet connection, otherwise 0.

Parameters

connection	

Implements **cs.usense.pipelines.mobility.tasks.ConnectionTask.ConnectionInterface** (p. 15).

6.21.3.2 donwloadTime()

```
void cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.donwloadTime (
            float networkUtilization )
```

This method is used to notify the network utilization calculated by this task.

Parameters

networkUtilization	

Implements **cs.usense.pipelines.mobility.tasks.DownloadTask.DonwnloadTaskInterface** (p. 18).

6.21.3.3 isComputing()

```
boolean cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.isComputing ( )
```

6.21.3.4 networkFinder()

```
void cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.networkFinder (
            int devices )
```

This method is used to notify when the task ends.

Parameters

devices	Number of devices connected to the AP.

Implements **cs.usense.pipelines.mobility.tasks.PingNetworkTask.FindOnNetworkInterface** (p. 21).

6.21.3.5 setIsComputing()

```
void cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.setIsComputing (
            boolean computing )
```

6.21.3.6 startRankingCalulation()

```
void cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.startRankingCalulation (
            String accessPointIp,
             MTrackerAP ap )
```

This function starts a new ranking coputaion of active paramiters.

Parameters

access←↩ PointIp	Ip of the acces point actually connected.
ap	MTracker access point connected.

6.21.4 Member Data Documentation

6.21.4.1 COMPUTE_FUNCTION_1

```
final boolean cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.COMPUTE_FUNCTIO←↩
N_1 = false
```

Function to be used in the computation.

6.21.4.2 COMPUTE_FUNCTION_2

```
final boolean cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.COMPUTE_FUNCTIO←↩
N_2 = false
```

Function to be used in the computation.

6.21.4.3 COMPUTE_FUNCTION_3

```
final boolean cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.COMPUTE_FUNCTIO←↩
N_3 = false
```

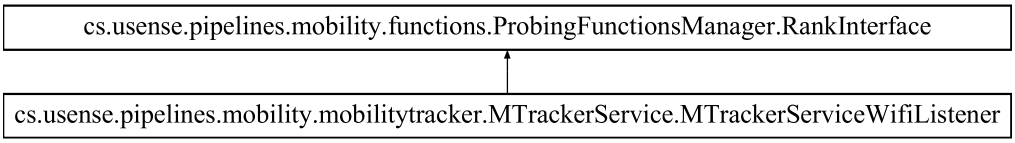
Function to be used in the computation.

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/functions/ **ProbingFunctionsManager.java**

6.22 cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.RankInterface Interface Reference

Inheritance diagram for cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.RankInterface:



Public Member Functions

- void **rank** (double rank, int function, **MTrackerAP** ap)

6.22.1 Detailed Description

This interface is used to notify when a rank function has a result.

6.22.2 Member Function Documentation

6.22.2.1 rank()

```
void cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.RankInterface.rank (
            double rank,
            int function,
            MTrackerAP ap )
```

This funtion is used to notify when a rank function has a result.

Parameters

rank	Rank value.
function	Funtion used to calcule the rank.
ap	Access point information.

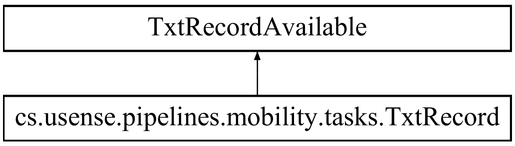
Implemented in **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener** (p. 57).

The documentation for this interface was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/functions/ **ProbingFunctionsManager.java**

6.23 cs.usense.pipelines.mobility.tasks.TxtRecord Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.TxtRecord:



Public Member Functions

- **TxtRecord** ()
- int **getmBestAPShared** ()
- double **getSumRank3** ()
- double **getSumRank4** ()
- Map **getMapSumRank4** ()
- void **setmBSSIDConnected** (String BSSID)
- void **deleteRecommendations** ()
- void **onTxtRecordAvailable** (String fullDomainName, Map< String, String > txtRecordMap, WifiP2pDevice srcDevice)

6.23.1 Detailed Description

Created by copelabs on 11/12/2017.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 TxtRecord()

```
cs.usense.pipelines.mobility.tasks.TxtRecord.TxtRecord ( )
```

6.23.3 Member Function Documentation

6.23.3.1 deleteRecommendations()

```
void cs.usense.pipelines.mobility.tasks.TxtRecord.deleteRecommendations ( )
```

6.23.3.2 getMapSumRank4()

```
Map cs.usense.pipelines.mobility.tasks.TxtRecord.getMapSumRank4 ( )
```

6.23.3.3 getmBestAPShared()

```
int cs.usense.pipelines.mobility.tasks.TxtRecord.getmBestAPShared ( )
```

6.23.3.4 getSumRank3()

```
double cs.usense.pipelines.mobility.tasks.TxtRecord.getSumRank3 ( )
```

6.23.3.5 getSumRank4()

```
double cs.usense.pipelines.mobility.tasks.TxtRecord.getSumRank4 ( )
```

6.23.3.6 onTxtRecordAvailable()

```
void cs.usense.pipelines.mobility.tasks.TxtRecord.onTxtRecordAvailable (
            String fullDomainName,
            Map< String, String > txtRecordMap,
            WifiP2pDevice srcDevice )
```

6.23.3.7 setmBSSIDConnected()

```
void cs.usense.pipelines.mobility.tasks.TxtRecord.setmBSSIDConnected (
            String BSSID )
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **TxtRecord.java**

6.24 cs.usense.pipelines.mobility.utils.Utils Class Reference

Static Public Member Functions

- static float **batteryStatus** (Context context)
- static void **setAlarm** (Context context, int hour, int minute)

6.24.1 Detailed Description

Created by copelabs on 16/10/2017.

6.24.2 Member Function Documentation

6.24.2.1 batteryStatus()

```
static float cs.usense.pipelines.mobility.utils.Utils.batteryStatus (
             Context context )  [static]
```

6.24.2.2 setAlarm()

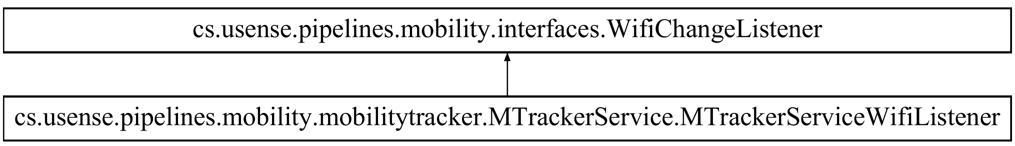
```
static void cs.usense.pipelines.mobility.utils.Utils.setAlarm (
             Context context,
             int hour,
             int minute )  [static]
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/utils/ **Utils.java**

6.25 cs.usense.pipelines.mobility.interfaces.WifiChangeListener Interface Reference

Inheritance diagram for cs.usense.pipelines.mobility.interfaces.WifiChangeListener:



Public Member Functions

- void **onWifiStateDisabled** (boolean valid, String bssid, String ssid, long visitId, long connectionStart, long connectionEnd)
- void **onWifiStateEnabled** ()
- void **onWifiConnectionDown** (boolean valid, String bssid, String ssid, long visitId, long connectionStart, long connectionEnd)
- long **onWifiConnectionUp** (String bssid, String ssid, List< ScanResult > lastScanResults)
- void **onWifiAvailableNetworksChange** (String bssid, List< ScanResult > results)
- void **onWifiAvailableList** (List< ScanResult > results)
- void **onConnectionRejected** (String bssid, String ssid)

6.25.1 Member Function Documentation

6.25.1.1 onConnectionRejected()

```
void cs.usense.pipelines.mobility.interfaces.WifiChangeListener.onConnectionRejected (
            String bssid,
            String ssid )
```

Implemented in **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener** (p. 55).

6.25.1.2 onWifiAvailableList()

```
void cs.usense.pipelines.mobility.interfaces.WifiChangeListener.onWifiAvailableList (
            List< ScanResult > results )
```

Implemented in **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener** (p. 55).

6.25.1.3 onWifiAvailableNetworksChange()

```
void cs.usense.pipelines.mobility.interfaces.WifiChangeListener.onWifiAvailableNetworksChange
(
            String bssid,
            List< ScanResult > results )
```

Implemented in **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener** (p. 55).

6.25.1.4 onWifiConnectionDown()

```
void cs.usense.pipelines.mobility.interfaces.WifiChangeListener.onWifiConnectionDown (
            boolean valid,
            String bssid,
            String ssid,
            long visitId,
            long connectionStart,
            long connectionEnd )
```

Implemented in **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener** (p. 56).

6.25.1.5 onWifiConnectionUp()

```
long cs.usense.pipelines.mobility.interfaces.WifiChangeListener.onWifiConnectionUp (
            String bssid,
            String ssid,
            List< ScanResult > lastScanResults )
```

Implemented in **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener** (p. 56).

6.25.1.6 onWifiStateDisabled()

```
void cs.usense.pipelines.mobility.interfaces.WifiChangeListener.onWifiStateDisabled (
            boolean valid,
            String bssid,
            String ssid,
            long visitId,
            long connectionStart,
            long connectionEnd )
```

Implemented in **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener** (p. 56).

6.25.1.7 onWifiStateEnabled()

```
void cs.usense.pipelines.mobility.interfaces.WifiChangeListener.onWifiStateEnabled ( )
```

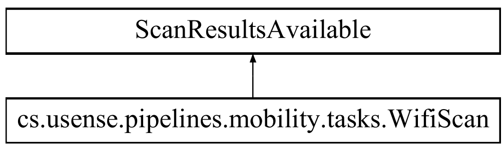
Implemented in **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener** (p. 56).

The documentation for this interface was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/interfaces/ **WifiChangeListener.java**

6.26 cs.usense.pipelines.mobility.tasks.WifiScan Class Reference

Inheritance diagram for cs.usense.pipelines.mobility.tasks.WifiScan:



Public Member Functions

- **WifiScan** ()
- void **onScanResultsAvailable** (List< ScanResult > scanResults)

6.26.1 Constructor & Destructor Documentation

6.26.1.1 WifiScan()

```
cs.usense.pipelines.mobility.tasks.WifiScan.WifiScan ( )
```

6.26.2 Member Function Documentation

6.26.2.1 onScanResultsAvailable()

```
void cs.usense.pipelines.mobility.tasks.WifiScan.onScanResultsAvailable (
            List< ScanResult > scanResults )
```

The documentation for this class was generated from the following file:

- NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ **WifiScan.java**

Chapter 7

File Documentation

7.1 NSense/app/src/main/java/cs/usense/pipelines/mobility/fragments/Attractiveness↩ DialogFragment.java File Reference

Classes

- class **cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment**
- interface **cs.usense.pipelines.mobility.fragments.AttractivenessDialogFragment.Attractiveness↩ DialogListener**

Packages

- package **cs.usense.pipelines.mobility.fragments**

7.2 NSense/app/src/main/java/cs/usense/pipelines/mobility/functions/Functions.java File Reference

Classes

- class **cs.usense.pipelines.mobility.functions.Functions**

Packages

- package **cs.usense.pipelines.mobility.functions**

7.3 NSense/app/src/main/java/cs/usense/pipelines/mobility/functions/ProbingFunctions↩ Manager.java File Reference

Classes

- class **cs.usense.pipelines.mobility.functions.ProbingFunctionsManager**
- interface **cs.usense.pipelines.mobility.functions.ProbingFunctionsManager.RankInterface**

Packages

- package **cs.usense.pipelines.mobility.functions**

7.4 NSense/app/src/main/java/cs/usense/pipelines/mobility/helpers/MTrackerData↩ Source.java File Reference

Classes

- class **cs.usense.pipelines.mobility.helpers.MTrackerDataSource**

Packages

- package **cs.usense.pipelines.mobility.helpers**

7.5 NSense/app/src/main/java/cs/usense/pipelines/mobility/helpers/MTrackerSQLite↩ Helper.java File Reference

Classes

- class **cs.usense.pipelines.mobility.helpers.MTrackerSQLiteHelper**

Packages

- package **cs.usense.pipelines.mobility.helpers**

7.6 NSense/app/src/main/java/cs/usense/pipelines/mobility/interfaces/DataBaseChange↩ Listener.java File Reference

Classes

- interface **cs.usense.pipelines.mobility.interfaces.DataBaseChangeListener**

Packages

- package **cs.usense.pipelines.mobility.interfaces**

7.7 NSense/app/src/main/java/cs/usense/pipelines/mobility/interfaces/WifiChange↩ Listener.java File Reference

Classes

- interface **cs.usense.pipelines.mobility.interfaces.WifiChangeListener**

Packages

- package **cs.usense.pipelines.mobility.interfaces**

7.8 NSense/app/src/main/java/cs/usense/pipelines/mobility/mobilitytracker/MTracker↩ Application.java File Reference

Classes

- class **cs.usense.pipelines.mobility.mobilitytracker.MTrackerApplication**

Packages

- package **cs.usense.pipelines.mobility.mobilitytracker**

7.9 NSense/app/src/main/java/cs/usense/pipelines/mobility/mobilitytracker/MTracker↩ Service.java File Reference

Classes

- class **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService**
- class **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.LocalBinder**
- class **cs.usense.pipelines.mobility.mobilitytracker.MTrackerService.MTrackerServiceWifiListener**

Packages

- package **cs.usense.pipelines.mobility.mobilitytracker**

7.10 NSense/app/src/main/java/cs/usense/pipelines/mobility/mobilitytracker/MTracker↩ WifiManager.java File Reference

Classes

- class **cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager**
- class **cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.WifiStateChange**
- class **cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.WifiConnectionChange**
- class **cs.usense.pipelines.mobility.mobilitytracker.MTrackerWifiManager.WifiAvailableNetworks↩ Change**

Packages

- package **cs.usense.pipelines.mobility.mobilitytracker**

7.11 NSense/app/src/main/java/cs/usense/pipelines/mobility/models/MTrackerAP.java File Reference

Classes

- class **cs.usense.pipelines.mobility.models.MTrackerAP**

Packages

- package **cs.usense.pipelines.mobility.models**

7.12 NSense/app/src/main/java/cs/usense/pipelines/mobility/models/MTrackerVisit.java File Reference

Classes

- class **cs.usense.pipelines.mobility.models.MTrackerVisit**

Packages

- package **cs.usense.pipelines.mobility.models**

7.13 NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/ConnectionTask.java File Reference

Classes

- class **cs.usense.pipelines.mobility.tasks.ConnectionTask**
- interface **cs.usense.pipelines.mobility.tasks.ConnectionTask.ConnectionInterface**

Packages

- package **cs.usense.pipelines.mobility.tasks**

7.14 NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/DownloadTask.java File Reference

Classes

- class **cs.usense.pipelines.mobility.tasks.DownloadTask**
- interface **cs.usense.pipelines.mobility.tasks.DownloadTask.DonwnloadTaskInterface**

Packages

- package **cs.usense.pipelines.mobility.tasks**

7.15 NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/PeerList.java File Reference

Classes

- class **cs.usense.pipelines.mobility.tasks.PeerList**

Packages

- package **cs.usense.pipelines.mobility.tasks**

7.16 NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/PingNetworkTask.java File Reference

Classes

- class **cs.usense.pipelines.mobility.tasks.PingNetworkTask**
- interface **cs.usense.pipelines.mobility.tasks.PingNetworkTask.FindOnNetworkInterface**

Packages

- package **cs.usense.pipelines.mobility.tasks**

7.17 NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/TxtRecord.java File Reference

Classes

- class **cs.usense.pipelines.mobility.tasks.TxtRecord**

Packages

- package **cs.usense.pipelines.mobility.tasks**

7.18 NSense/app/src/main/java/cs/usense/pipelines/mobility/tasks/WifiScan.java File Reference

Classes

- class **cs.usense.pipelines.mobility.tasks.WifiScan**

Packages

- package **cs.usense.pipelines.mobility.tasks**

7.19 NSense/app/src/main/java/cs/usense/pipelines/mobility/utils/Utils.java File Reference

Classes

- class **cs.usense.pipelines.mobility.utils.Utils**

Packages

- package **cs.usense.pipelines.mobility.utils**

Index