Action full title:

Universal, mobile-centric and opportunistic communications architecture

Action acronym:

UMOBILE

Deliverable:

D3.1 - UMOBILE architecture report (1)

Project Information:

Project Full Title	Universal, mobile-centric and opportunistic communications architecture
Project Acronym	UMOBILE
Grant agreement number	645124
Call identifier	H2020-ICT-2014-1
Topic	ICT-05-2014 Smart Networks and novel Internet Architectures
Programme	EU Framework Programme for Research and Innovation HORIZON 2020
Project Coordinator	Prof. Vassilis Tsaoussidis, Democritus University of Thrace

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

Deliverable Information:

Deliverable Number-Title	D3.1 - UMOBILE architecture report (1)
WP Number	3
WP Leader	DUTH
Task Leader(s)	DUTH
Contributing Partners	DUTH, UCL, UCAM, COPELABS, TECNALIA, SENCEPTION
Authors	DUTH: Ioannis Komnios, Sotiris Diamantopoulos, Vassilis Tsaoussidis UCL: Sergi Rene, Ioannis Psaras UCAM: Adisorn Lertsinsrubtavee COPELABS: Paulo Mendes, Waldir Moreira, Luis Amaral Lopes TECNALIA: Iñigo Sedano, Susana Sanchez Perez SENCEPTION: Rute Sofia
Contact	ikomnios@ee.duth.gr
Due date	31/05/2016
Actual date of submission	31/05/2016

Dissemination level:

PU	Public	X
CO	Confidential, only for members of the consortium (including the Commission Services)	
CI	Classified, as referred to in Commission Decision 2001/844/EC	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

Executive Summary

Background: This report is written in the framework of Task 3.1 “DTN overlay design and convergence layers for underlying protocols” of UMOBILE project. The deliverable aims to describe work on the UMOBILE core architecture and is accompanied by the implementation of the architecture so far and documentation on the code.

Objectives: The core activity of WP3 is the design and implementation of the UMOBILE platform. Departing from the existing properties of DTN and ICN, we establish an architectural framework that extends connectivity options by being delay-tolerant and exposing a common information-centric abstraction to applications.

UMOBILE aims to advance networking technologies and architectures towards the conception and realization of Future Internet. In particular, UMOBILE extends Internet (i) functionally – by combining ICN and DTN technologies within a new architecture, (ii) geographically – by allowing for internetworking on demand over remote and isolated areas – and (iii) socially – by allowing low-cost access and free user-to-user networking.

The goal of this document is to provide a detailed description, along with a manual when applicable, of the new features, mechanisms and applications that have been developed so far as part of the UMOBILE architecture. Implementation code so far is also provided. The basis for UMOBILE platform is the Named Data Networking (NDN) architecture, one of the most promising ICN implementations, and UMOBILE features are being built in line with NDN. In this deliverable:

1. We first describe the UMOBILE vision and highlight our contributions in each part of the architecture,
2. We then focus on the specific features that we have designed as part of UMOBILE and
3. We conclude with a clear description of the next steps that the consortium will follow to achieve the integration of all components and the extensive evaluation of UMOBILE platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

List of Definitions

Term	Meaning
AP	An access point (AP) is a networking hardware device that allows a Wi-Fi compliant device to connect to a wired network.
Application	Computer software design to perform a single or several speciBic tasks, e.g. a calendar and map services. In UMOBILE, context-aware applications are considered.
BER	In digital transmission, the number of bit errors is the number of received bits of a data stream over a communication channel that have been altered due to noise, interference, distortion or bit synchronization errors. The bit error rate (BER) is the number of bit errors per unit time.
BP	Bundle Protocol (BP) defines the bundle as the core unit of the DTN architecture; a bundle is a series of data blocks that is routed in a store-and-forward manner between nodes over various transport networks.
BT	Bluetooth is a wireless technology standard for exchanging data over short distances from fixed and mobile devices, and building personal area networks (PANs).
CIT	Carried Interest Table is responsible for keeping up-to-date information concerning the data interests of the current node along with its social weights towards other nodes with whom it socially interacts.
CM	The Content Manager (CM) module in Oi! is responsible to manage all the messages to be sent, as well as received messages.
Content	Content refers to a piece of digital information that is disseminated and consumed by the end user equipment.
CS	A temporary cache of Data packets the router has received. Caching NDN Data packet helps satisfy future Interests for the same data faster. Various replacement strategies are implemented for the content store.
Data	Data is raw. Data is numbers that have no interpretation. In UMOBILE, data is the output of sensors (e.g. positions, activity, sound, wireless proximity).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

Data packet	In NDN, once the Interest reaches a node that has the requested data, the node will return a Data packet that contains both the name and the content, together with a signature by the producer's key which binds the two. This Data packet follows in reverse the path taken by the Interest to get back to the requesting consumer.
DM	Decision Maker is responsible for deciding whether replication should occur based on the level of social interaction towards specific interests, based on the SCORP algorithm.
DTN	Delay Tolerant Networking (DTN) supports interoperability of other networks by accommodating long disruptions and delays between and within those networks. DTN operates in a store-and-forward fashion where intermediate node can temporarily keep the messages and opportunistically forward them to the next hop. This inherently deals with temporary disruptions and allows connecting nodes that would otherwise be disconnected in space at any point in time by exploiting time-space paths.
EC	European Commission
E2E	In networks designed according to the end-to-end (E2E) principle, application-specific features reside in the communicating end nodes of the network, rather than in intermediary nodes, such as gateways, that exist to establish the network.
FIB	A routing table which maps name components to interfaces. The FIB itself is populated by a name-prex based routing protocol, and can have multiple output interfaces for each prex.
FN	Forwarding Node is responsible for routing requests for services towards the available copies in the service migration module.
Gateway	Gateway typically means an equipment installed at the edge of a network. It connects the local network to larger network or Internet. In addition, gateway also has the capability to store services and contents in its cache to subsequently provide localized access.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

IBR-DTN	IBR-DTN is a framework for DTN applications; its module-based architecture with miscellaneous interfaces makes it possible to change functionalities like routing or bundle storage just by inheriting a specific class.
ICN	Information-Centric Networking (ICN) supports efBicient delivery of both content and services by identifying information by name rather than the actual location. This decoupling of the information from its actual location breaks the need for end to end connectivity thus enabling much wider Blexibility for efficient content and service retrieval. ICN also inherently supports caching thus enabling much better localised communications.
Information	Information is about understanding what the data is telling us. It provides an understanding about what is happening to users so we can make it easier for them to get the content they need when they need it. In UMOBILE, information is the output of inference processes (e.g. affinity networks, roaming patterns, social isolation, crowd detection), which maybe performed over different types of data (data fusion).
Interest	A parameter capable of providing a measure (cost) of the “attention” of a user towards a specific content in a specific time instant. Users can cooperate and share their interests.
Interest packet	In NDN, a consumer puts the name of a desired piece of data into an Interest packet and sends it to the network. Routers use this name to forward the Interest toward the data producer(s).
NACK	A negative-acknowledge character (NAK or NACK) is a transmission control character sent by a station as a negative response to the station with which the connection has been set up.
NDN	Named Data Networking (NDN) is a Future Internet architecture that aims to transition today’s host-centric network architecture into a data-centric network architecture. In particular, users will no longer need to retrieve data from a specific physical location; instead, users will be able to search for content, independent of the location where the content is stored.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

NDN-CXX	NDN-CXX library (NDN C++ library with eXperimental eXtensions) provides the various common services shared between different NDF module.
NFD	The NDN Forwarding Deamon
Node	A wireless or wired capable device.
OPEX	An operating expense (OPEX) is an ongoing cost for running a product, business, or system. Its counterpart, a capital expenditure (CAPEX), is the cost of developing or providing non-consumable parts for the product or system.
OS	An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs. The operating system is a component of the system software in a computer system. Application programs usually require an operating system to function.
PerSense	PerSense is an open-source sensing platform that senses the environment of a user and provides relevant services.
PIT	A table that stores all the Interests that a router has forwarded but not satised yet. Each PIT entry records the data name carried in the Interest, its incoming and outgoing interface.
RT	RT is the routing module in Oi! App
RTT	Round-trip time (RTT) is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received. This time delay therefore consists of the propagation times between the two points of a signal.
SC	Service Controller manages the mapping of publishers of services and subscribers of services in the service migration module.
SEG	Service Execution Gateway is the point of attachment for clients in the the service migration module.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

Service	Service refers to a computational operation or application running on the network which can fulBill an end user's request. The services can be hosted and computed in some speciBic nodes such as servers or gateways. SpeciBically, services are normally provided for remuneration, at a distance, by electronic means and at the individual request of a recipient of services. For the purposes of this deBinition;"at a distance" means that the service is provided without the parties being simultaneously present; "by electronic means" means that the service is sent initially and received at its destination by means of electronic equipment for the processing (including digital compression) and storage of data, and entirely transmitted, conveyed and received by wire, by radio, by optical means or by other electromagnetic means; "at the individual request of a recipient of services" means that the service is provided through the transmission of data on individual request. Refer to D2.2 for further details.
SLA	Service-level agreement, a contractual agreement on the level of service to be provided by a service provider to a customer.
Social trust	Trust which builds upon associations of nodes is based on the notion of shared interests; individual or collective expression of interests; afBinities between end users.
SP	Service Publisher refers to the original content producer in the the service migration module.
SW	Social Weight (SW) measures the level of interaction between nodes.
SWCDG	Social Weight and Carried Data Gatherer is responsible for obtaining the list of interests and social weights towards the encountered node.
SWM	Social Weight Measurer is responsible for keeping track of the contact duration between the current and encountered nodes.
SWR	Social Weight Repository is responsible for storing the list of interests the current node comes across (obtained upon encountering a peer).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

tCDT	Temporary Carried Data Table holds information about the data that the encountered node is currently carrying.
TCP	The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network.
TECD	Time-Evolving Contact Duration
TLV	Type-Length-Value (TLV) is an encoding format used in NDN.
tPIT	temporary Pending Interest Table holds the list of interests seen by the encountered node and the social weights between itself and such interests.
TR	Technical Report
TTL	Time to live (TTL) is a mechanism that limits the lifespan or lifetime of data in a computer or network. TTL may be implemented as a counter or timestamp attached to or embedded in the data. Once the prescribed event count or timespan has elapsed, data is discarded.
UAV	Unmanned Aerial Vehicle, which is an aircraft with no pilot on board.
UDP	The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.
UI	User Interface (UI) module in Oi! allows the user to compose a message, choose its recipient from a list of known contacts, and to see received messages.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

UMOBILE architecture	A mobile-centric service-oriented architecture that efBiciently delivers content and services to the end users. The UMOBILE architecture integrates the principles of both DTN and ICN to enable reliable delivery of both content and services to the end users.
UMOBILE system	UMOBILE System refers to an open system that provides communication access to users through wired or wireless connectivity. This system exploits the beneBit of local communication to minimize upstream and downstream trafBic. The service or content can be exchanged and stored in several devices such as gateways; user equipment; customer premises equipment such as Wi-Fi Access Points in order to efBiciently deliver the desired contents or services to end users.
User	An entity (individual or collective) that is both a consumer and a relay of user services.
User equipment	User-equipment (UE) corresponds to a generic user terminal (for example a smart phone or notebook). In terms of UE and for operating systems we consider mainly smartphones equipped with Android; notebooks with UNIX, Windows, Mac OS.
User requirement	User requirement corresponds to the speciBications that users expect from the application, device or network.
User service	Context-aware services are considered as a set of mechanisms that assist incorporating information about the current surrounding of mobile users in order to provide more relevant of services.
User-centric	User-centric refers to a new paradigm leveraging user information at large to deliver novel content or services by users towards other users.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

1 Introduction

The main objective of UMOBILE is to develop a mobile-centric service oriented architecture that efficiently delivers content/services to the end-users. UMOBILE decouples services from their origin locations, shifting the host-centric paradigm to a new paradigm, one that incorporates aspects from both information-centric and opportunistic networking with the ultimate purpose of delivering an architecture focused in: i) improving aspects of the existing infrastructure (e.g., keeping traffic local to lower OPEX); ii) improving the social routine of Internet users via technology-mediated approaches; iii) extending the reach of services to areas with little or no infrastructure (e.g., remote areas, emergency situations).

UMOBILE aims to push network services (e.g., mobility management, intermittent connectivity support) and user services (e.g., pervasive content management) as close as possible to the end-users. By pushing such services closer to the users, we can optimize, in a scalable way, aspects such as bandwidth utilization and resource management. We can also improve the service availability in challenged network environments. For example, users in some areas may suffer from intermittent and unstable Internet connectivity when they are trying to access the services.

To achieve this, the proposed UMOBILE architecture combines two emerging architecture and connectivity approaches: Information Centric Networking (ICN) and Delay Tolerant Networking (DTN). The aim is to build an architecture that defines a new service abstraction that brings together both information centric as well as delay tolerant networking principles into one single abstraction.

In this document, we describe in detail the UMOBILE architecture that:

- Integrates ICN and DTN in order to provide delay-tolerant features in an information-centric framework,
- Includes newly developed apps (such as a chat app) that achieve opportunistic data transmission directly between users without the need for an access point (AP),
- Supports the migration of services at the edges of the network and closer to the end-users, so that services can be accessed faster, even in cases of disaster when typical Internet connectivity fails,
- Includes a smart routing mechanism that is adaptive to different communication scenarios,
- Introduces push services in the NDN architecture that is inherently based on the pull model,
- Uses a tool that captures wireless footprint aspects of end-users to assist in usage contextualisation,
- Supports an innovative naming scheme that adds new features to UMOBILE through the use of hashtags and
- Enables a new application-centric information sharing framework oriented to support and provide opportunistic computing to mobile devices (smartphones, tablets, etc.).

In addition to the description of the aforementioned components of the UMOBILE architecture, the present deliverable also includes:

- Code so far on the integration of NDN and IBR-DTN (manual is included in the deliverable);

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

- Code on the migration of a simple web server;
- Code of Oi! application (available in GitHub);
- Code of SOCIO application (available in GitHub);
- Guidlines to download and test PerSense Mobile Light service, and
- Technical report on Oi! and SOCIO (attached to D3.1 as Annex 1).

The document is organised as follows:

- Section 2 details the starting points for the UMOBILE architecture,
- Section 3 provides an insight to the vision of UMOBILE and highlights how each UMOBILE feature contributes to this vision,
- Section 4 describes the architectural status of the UMOBILE architecture and the progress so far regarding the integration of NDN and IBR-DTN,
- Section 5 details the new services, mechanisms and apps that have been developed as part of the UMOBILE platform, along with a manual where applicable,
- Section 6 outlines the next steps towards the integration of the aforementioned UMOBILE components and
- Section 7 concludes the document.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

2 UMOBILE starting points

In this Section, we start by providing a brief description of the two connectivity approaches that constitute the basis for UMOBILE architecture (Section 2.1), namely Named Data Networking (NDN) and Delay-Tolerant Networking (DTN). We then briefly refer to the end-user, system and network requirements that the consortium has defined for UMOBILE in D2.1 and D2.2 (Section 2.2).

2.1 Existing frameworks

The aim of UMOBILE is to build an architecture that defines a new service abstraction, bringing together information centric and delay tolerant networking principles into one single abstraction. To achieve that, UMOBILE uses two of the most promising ICN and DTN implementations as starting points: Named Data Networking and Delay-Tolerant Networking.

2.1.1 Named Data Networking (NDN)

Named Data Networking (NDN) is a Future Internet architecture that aims to transition today's host-centric network architecture into a data-centric network architecture. In particular, users will no longer need to retrieve data from a specific physical location; instead, users will be able to search for content, independent of the location where the content is stored. NDN changes the semantics of network service from delivering the packet to a given destination address to fetching data identied by a given name; NDN is a part of the broader information-centric networking approaches. In ICN approaches, data becomes independent from location, application, storage, and means of transportation, enabling in-network caching and replication.

Our initial approach was to introduce a "content layer" that intercepts communication, produces unique location-independent names for requested content and stores the latter within the network according to sophisticated caching policies. This way, the first stage of communication between the user and the content source, i.e., the content resolution stage, follows the approach of the current Internet and uses search engines, with the URL containing the name of the primary content server, while later stages of communication are based on the location-independent names for requested content.

However, in the UMOBILE project we focus on the mobile part of the network, where Internet nodes are mobile and connectivity can be disrupted, e.g., in an emergency scenario where the network might get fragmented and communication takes place in an ad hoc manner between mobile devices. In this scenario, we have to overcome the difficulties of a host-centric and connection-oriented communication paradigm. That said, we have come to realise that our initial plans for operating on top of an extra content-layer would not provide huge benefits, but would rather recycle old problems with regard to mobility support.

Thus, we decided to rely on pure ICN features to develop our UMOBILE architecture. Such features include flexibility, host multihoming, content name consistency and resiliency and provide a solid base to build an architecture for mobile, infrastructureless or delay tolerant networks.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

Among all ICN architectures, we decided to use NDN as the baseline. NDN is the most promising ICN architecture with more participants in the ICN community, and with more active projects implementation-wise. We think NDN is a perfect candidate as a starting point for our UMOBILE architecture. However we need to improve NDN scalability by providing a new keyword-based naming system.

To provide compatibility with existing mobile user devices, we devise the UMOBILE architecture as a layer working on top of IP, where IP is used as a network enabler but only on a hop-by-hop basis and is linked to the wireless connectivity. Also, in order to provide connectivity between any opportunistic UMOBILE device and the fixed network (IP network), we devise a UMOBILE Proxy/Gateway able to translate interest packets to HTTP requests and vice versa.

Communication in NDN is driven by receivers i.e., data consumers, through the exchange of two types of packets: Interest and Data. Both types of packets carry a name that identies a piece of data that can be transmitted in one Data packet.

- **Interest**: A consumer puts the name of a desired piece of data into an Interest packet and sends it to the network. Routers use this name to forward the Interest toward the data producer(s).
- **Data**: Once the Interest reaches a node that has the requested data, the node will return a Data packet that contains both the name and the content, together with a signature by the producer's key which binds the two. This Data packet follows in reverse the path taken by the Interest to get back to the requesting consumer.

The core of NDN architecture lies in a network forwarder, the NDN Forwarding Daemon (NFD). NFD is a modular and extensible forwarder that implements the forwarding of both Interest and Data packets. To achieve that, it abstracts lower-level network transport mechanisms into NDN Faces that provide the necessary abstraction on top of various lower level transport mechanisms.

Several data structures are used to support forwarding of NDN Interest and Data, the most important being:

- **Content Store (CS)**: A temporary cache of Data packets the router has received. Caching NDN Data packets helps satisfy future Interests for the same data faster. Various replacement strategies are implemented for the content store.
- **Pending Interest Table (PIT)**: A table that stores all the Interests that a router has forwarded but not satised yet. Each PIT entry records the data name carried in the Interest, its incoming and outgoing interface.
- **Forwarding Information Base (FIB)**: A routing table which maps name components to interfaces. The FIB itself is populated by a name-prex based routing protocol, and can have multiple output interfaces for each prex.

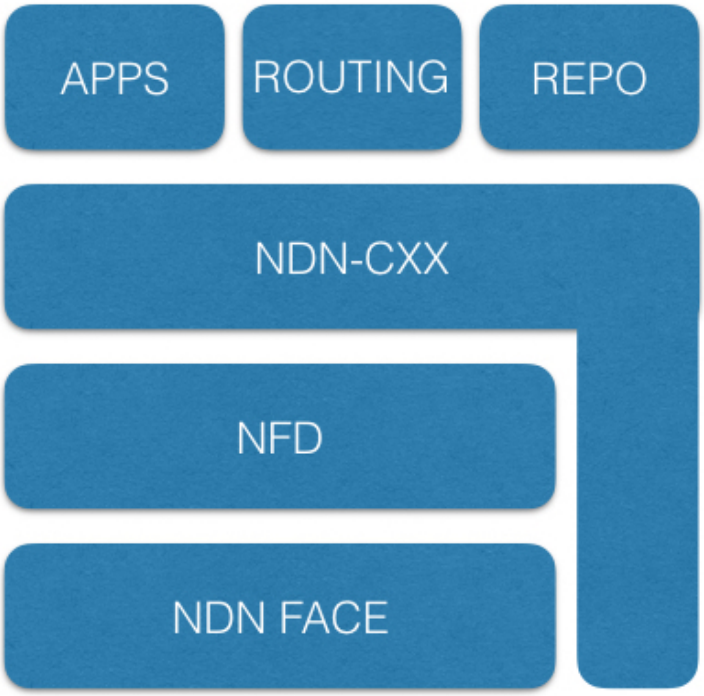
NDN-CXX library (NDN C++ library with eXperimental eXtensions) provides the various common services shared between different NDF module. On top of NDN-CXX, NDN supports:

- NDN Common Client libraries that provide APIs for Python, C++, Java and JavaScript,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

- A routing module that supports conventional routing algorithms such as link state and distance vector adapted to route on name prexes and
- A repository for persistent storage.

The following Figure depicts the high-level design of NDN architecture.



NDN architecture

More information on NDN architecture can be found on Named Data Networking website: `http://named-data.net`.

2.1.2 IBR-DTN Delay Tolerant Networking

DTN is an emerging technology to support a new era in internetworking and interoperable communications, either on Earth, or in Space. Like IP, DTN operates on top of existing network architectures, creating a DTN overlay. In particular, DTN extends internetworking in the time domain: rather than assuming a continuous end-to-end (E2E) path as IP networks do, DTN operates in a store-and-forward fashion: intermediate nodes assume temporary responsibility for messages and keep them until the next opportunity arises to forward them to the next hop. While stored, messages may even be physically carried within a node as the node is transported: the model is also termed store-carry-forward. This inherently deals with temporary disconnections or disruptions and allows the connection of nodes that would else be disconnected in space at any point in time.

Key design assumptions of Internet protocols such as short round-trip time (RTT), absence of disruptions and continuous E2E path availability are challenged by such a concept. In particular, DTN operates successfully over challenged networks that may be characterised by:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

- Disruptive connectivity, which means that a connection between the sender and the receiver is not always available (and may never be);
- High propagation delay, which ranges from a few seconds to several minutes;
- High bit error rates (BER) up to 10-3 for deep-space networks, and
- High bandwidth asymmetry, which can reach 1000:1 for space links.

The core of the DTN architecture lies in the Bundle Protocol (BP). BP defines the bundle as the core unit of the DTN architecture; a bundle is a series of data blocks that is routed in a store-and-forward manner between nodes over various transport networks. In order to improve the efficiency of transfers, the DTN architecture supports several features, such as fragmentation and custody transfer. DTN fragmentation and reassembly ensures that contact volumes are fully utilised, avoiding retransmission of partially-forwarded bundles. Bundle fragmentation can be performed either proactively at the sender or reactively at an intermediate node, if the reduction of the size of a bundle is required to forward it. Reassembly can be performed either at the destination or at some other node on the route to the destination. Custody transfer is another important feature of the DTN architecture. In essence, custody transfer moves the responsibility for reliable delivery of a bundle among different DTN nodes in the network. Whenever a node accepts the reliable delivery responsibility of a bundle, it is called the “custodian” of this bundle.

As far as the existing DTN reference implementations are concerned, several implementations have been developed during the last years, each targeting different applicability scenarios. IBR-DTN is one of the most popular, well-documented and maintained implementations of the bundle protocol designed for embedded systems. IBR-DTN can be used as framework for DTN applications; its module-based architecture with miscellaneous interfaces makes it possible to change functionalities like routing or bundle storage just by inheriting a specific class.

IBR-DTN runs on a variety of platforms spanning from Debian and Ubuntu to Android operating systems (OS) and supports not only Bundle Protocol, but also Bundle Security Protocol, the security extension of BP that provides integrity and confidentiality services. Therefore, IBR-DTN is a suitable candidate for UMOBILE since it can be used not only on stationary nodes, but also on mobile nodes that are the main focus of UMOBILE project. IBR-DTN supports a variety of convergence layer protocols, including:

- TCP/IP convergence layer;
- TLS extension for TCP convergence layer;
- UDP/IP convergence layer;
- IP neighbor discovery;
- HTTP convergence layer;
- IEEE 802.15.4 LoWPAN convergence layer and
- Generic datagram convergence layer with IEEE 802.15.4 and UDP support.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

More information on IBR-DTN can be found on the project's website: `https://www.ibr.cs.tu-bs.de/projects/ibr-dtn/`.

2.2 UMOBILE requirements

The implementation of UMOBILE is in line with the requirements defined in D2.1 and D2.2. In particular, the consortium defined four applicability scenarios for UMOBILE architectures and, based on them, defined:

- End-user requirements in different environments including urban, remote and disaster areas, and
- System and network requirements, as well as assumptions, for the high-level design of the UMOBILE architecture.

The overall UMOBILE requirements have been divided into three types:

- MUST, which are an absolute requirement of the UMOBILE specification, and so are required in any implementation;
- SHOULD, which are recommended features, meaning that there may exist valid reasons in particular circumstances to ignore such features, but the full implications must be understood and carefully weighed before choosing a different course, and
- MAY, which are truly optional features.

The initial requirements will be revised by M30, after UMOBILE architecture has been fully implemented.

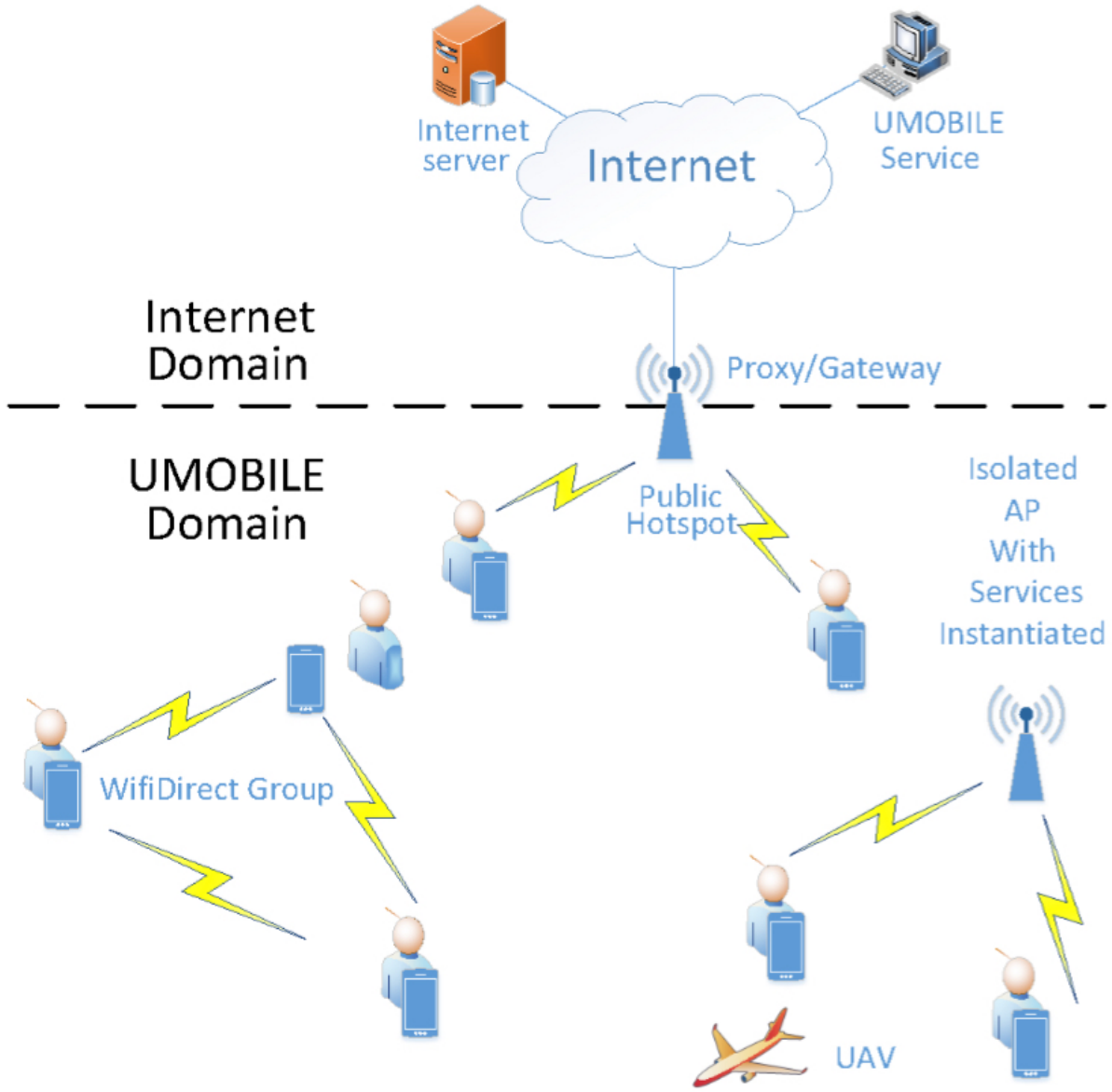
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

3 UMOBILE vision

In D3.3, we described the UMOBILE architecture and highlighted the key services that the proposed architecture will provide. The Figure below (extracted from D3.3) shows the high-level design of the UMOBILE architecture divided into two distinct domains: the UMOBILE domain and the Internet domain. The actors involved in the UMOBILE architecture include:

- UMOBILE-enabled mobile devices (i.e., smartphone, tablet), used to send and receive participatory data (e.g. photos, short messages) as well as opportunistic data (e.g. atmospheric pressure, temperature, noise, people anxiety levels, roaming patterns).
- UMOBILE-enabled hotspots able to collect and relay relevant information (e.g., alert messages, instructions from emergency authorities), host some instantiated services or store collected data, check its validity and perform computational functions (e.g. data fusion) to increase the value of the information to the civil authorities.
- UMOBILE-enabled UAV devices or other vehicles able to collect and relay relevant information and connect two isolated areas.
- UMOBILE-enabled gateways/proxies provides interconnectivity between UMOBILE domain and the Internet domain.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124



High-level design of UMOBILE architecture

The aim of UMOBILE project is to provide an architecture that merges information-centric networking with delay-tolerant networking in order to efficiently operate in different network situations, reaching disconnected environments and users and providing new types of services. These services are based on pull or push communication models and can be exploited from a variety of applications such as chat and local news applications. Having already described in detail the envisioned UMOBILE services in D3.3, in this deliverable we focus on the specific enhancements and modifications of the NDN platform that are required to provide these new services.

In particular, we build the UMOBILE architecture using the NDN architecture as our starting point and:

- Create two new applications: a chat app and a news app;
- Provide two new types of services that can be used in a variety of scenarios: a push service for cases when

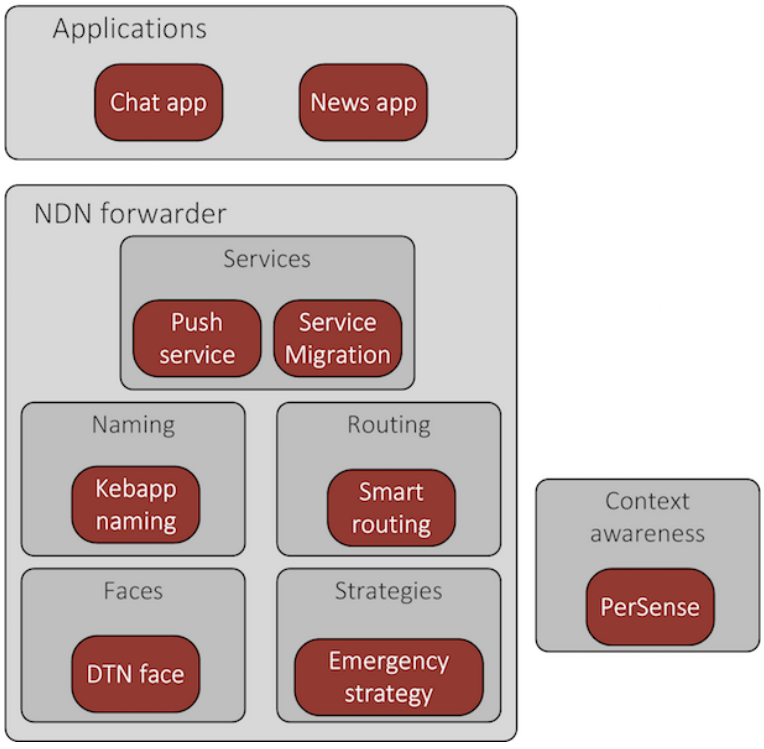
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

the user needs to push data to the network (e.g. in cases of emergency) and a service migration that can moves services closer to the edge of the network where they can be accessed significantly faster, even in cases when the remote server is not available;

- Develop a new smart routing framework that also takes into consideration user context and behavior;
- Build a new delay-tolerant face in the NDN architecture. This DTN face is used in opportunistic and disruptive scenarios and delivers packets to the IBR-DTN, which is then responsible for their delay-tolerant transmission;
- Develop an emergency strategy that allows for the broadcast of information in cases of emergency and
- Propose a new application-centric naming, where applications share common name-spaces and further support the use of a keywords.

As an additional tool, we also use PerSense, an open-source sensing platform, to sense the environment of a user and provide relevant services.

An overview of the aforementioned mechanisms that constitute the UMOBILE architecture is depicted below.

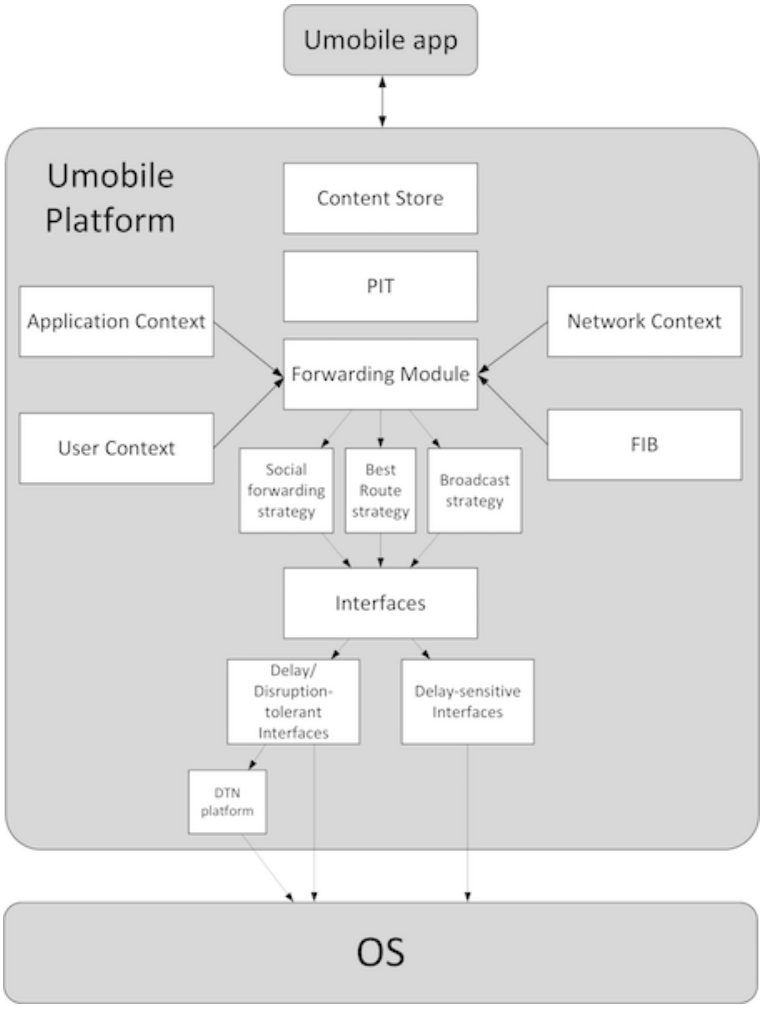


New mechanisms in UMOBILE

Architecturally, the UMOBILE architecture consists of the modules depicted in the Figure below. As shown, the newly-developed applications sit on top of the UMOBILE platform, which communicates with the OS of the device. The forwarding module uses input in terms of application, user and network context to select the most suitable forwarding strategy in each scenario. In particular, we develop a new social forwarding strategy that exploits social metrics to deliver data, as well as a broadcast strategy that allows for the delivery of the data to a large number

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

of users. Apart from the typical delay-sensitive interfaces, such as TCP, we also develop delay/disruption-tolerant interfaces that are used for delay-insensitive data transmissions or in case of remote environments and opportunistic users.



UMOBILE architecture modules

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

4 UMOBILE Architectural status

Having a clear view of the applicability scenarios of UMOBILE architecture (as described in D2.1), the system and network requirements (as described in D2.2) and the UMOBILE architecture design (as described in D3.3), the UMOBILE consortium has been working on the implementation of the new mechanisms and services described in the section above. This deliverable provides an insight in each of these mechanisms and details their implementation status. Along with the present report, we also submit the relevant code, even if not fully implemented yet. The full implementation of the UMOBILE architecture will be delivered to the EC on M30, as D3.2, according to the project's work plan.

4.1 Integration of NDN and IBR-DTN

4.1.1 Description

The main goal of UMOBILE project is the development of a networking architecture that supports both information-centricity and delay-tolerance. To achieve that, we exploit the Named Data Networking (NDN) platform and the IBR-DTN implementation, briefly described in Section 2. Information-centricity in inherently provided by NDN, while we aim to provide delay-tolerance in two different ways:

- Through DTN tunnelling in case of opportunistic communications when delay-tolerant transmission of data is deemed the most suitable alternative, or
- Extended timers to allow for longer interest lifetimes.

Below we detail how UMOBILE core architecture works in both cases.

DTN tunnelling NFD, as a network forwarder, moves packets between network interfaces. So far, NFD supports:

- Unix stream transport for stream-oriented Unix domain sockets;
- Ethernet transport to communicate directly over Ethernet;
- UDP unicast transport that communicates on UDP tunnels over IPv4 or IPv6;
- UDP multicast transport that communications on a UDP multicast group;
- TCP unicast transport that communicates on TCP tunnels over IPv4 or IPv6 and
- WebSocket transport, implementing a message-based protocol on top of TCP.

NFD provides information-centricity over a variety of interfaces, however, it cannot provide delay-tolerant characteristics. For this reason, we enhance NFD by developing the required protocol factories and faces to provide DTN tunnelling as well, using IBR-DTN. In particular, when the selection of the DTN face seems to be the best choice (e.g. in terms of cost), Interest packets are handled over to IBR-DTN in order to be encapsulated into DTN bundles.

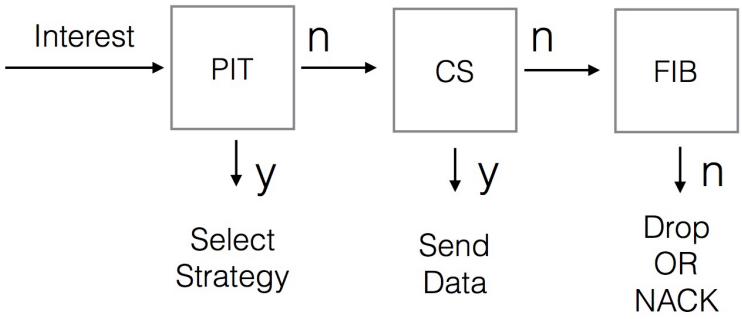
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

For example, we assume a Forward Information Base (FIB) table with two entries for the same name prefix, as shown below.

Name Prefix	(Face, Cost)
/map	(180, 5)
/map	(181, 100)

In the first case, we know that Face 180 corresponds to a UDP face with *remote interface udp://192.168.1.1:6363* and *local interface udp://192.168.1.72:6363*. In the second case, Face 181 corresponds to a DTN face with *remote interface dtn://node3* and local interface dtn://node1. The combination of (Face, Cost) is called NextHop and NextHop records are ordered by ascending cost within a FIB entry.

Now let's assume a scenario where we have three UMOBILE nodes in a disconnected environment (i.e. no node is in range with another), so no interface is available for transmission. When Node 1 issues an Interest, it first checks its Pending Interest Table (PIT) to see if there already exists an entry for the same Interest. If no entry exists, Node 1 checks whether the data requested by this Interest are stored in the Content Store (CS); this is possible in case another node recently issued the same Interest, which was delivered to the requester through Node 1. If a copy of the data exists in the CS, it is delivered to the applications of Node 1 that issued the Interest. If no copy is found in the CS, Node 1 checks the entries of the FIB table in order to check if a valid entry exists. Since the environment is disconnected and no interface is currently available, in a typical NDN scenario no entry would exist in the FIB table and, therefore, the Interest would be dropped and/or a NACK would be sent to the application that issued the Interest, as shown in the Figure below.



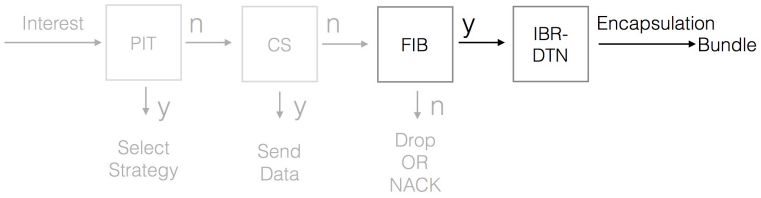
Interest operation at Node 1 (source node)

However, in UMOBILE an entry that corresponds to the DTN face exists in the FIB table to support operation in disconnected environments, as shown in the table below.

Name Prefix	(Face, Cost)
/map	(181, 100)

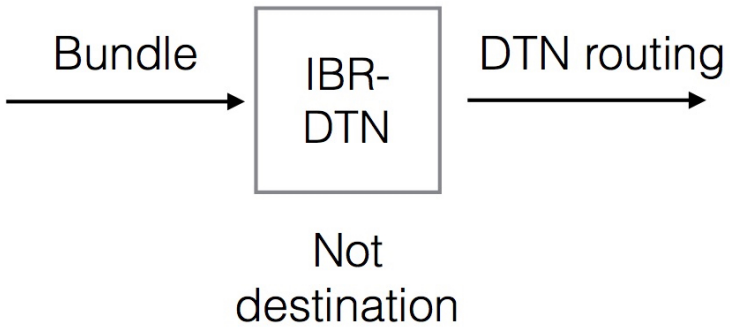
In this case, the Interest is handled from NFD to IBR-DTN, where it is encapsulated into a bundle. IBR-DTN is now responsible for the delivery of the encapsulated Interest to the next hop, based on the routing strategy that IBR-DTN follows. For example, if epidemic routing is selected, IBR-DTN in Node 1 will deliver the encapsulated Interest to all nodes that come into range. Of course, the destination node for the bundle is the entry of the remote interface in the FIB table, i.e. *dtn://node3*.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124



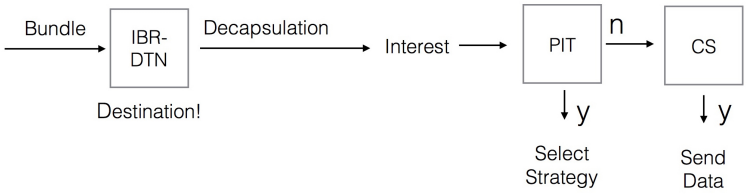
Interest encapsulation to bundle at Node 1 (source node)

After some time, Node 2 comes into range of Node 1 and a copy of the bundle is delivered to IBR-DTN of Node 2. IBR-DTN of Node 2 checks the destination address of the bundle and, since this is not the destination node, holds the bundle in its storage and further forwards copies of the bundle to any nodes it may encounter, as shown in the Figure below.



Operation at Node 2 (intermediate node)

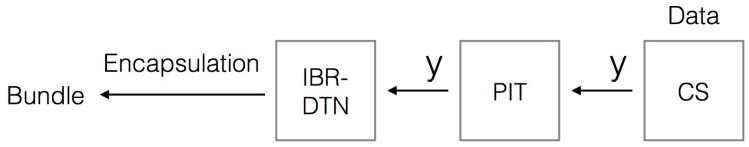
Assuming the Node 2 encounters Node 3 later on, the encapsulated Interest is delivered to the IBR-DTN of Node 3, which is the destination node for this bundle. Therefore, IBR-DTN of Node 3 decapsulates the Interest and delivers it to its NFD. The first step is to check if a PIT entry exists for this Interest. If not, the Interest arrives at the CS, where the requested Data packet is found and sent to Node 1 (which issued the Interest) following the reverse path.



Interest decapsulation and NFD operation at Node 3 (destination node)

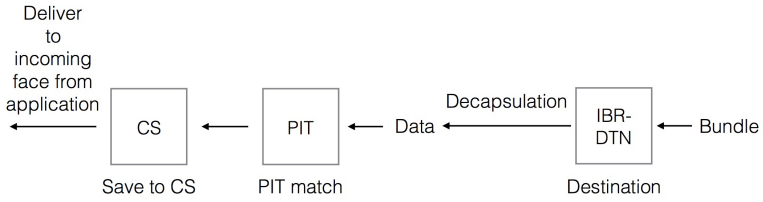
In particular, the Data packet leaves the content store and finds the relevant incoming face as an entry of the PIT table (in this case, *dtn://node1* for the remote interface). Then, the Data packet is handled over to the IBR-DTN of Node 3, which encapsulates it into a bundle with destination address *dtn://node1*. The bundle is then opportunistically forwarded until it reaches the source node.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124



Data encapsulation and transmission at Node 3 (destination node)

The IBR-DTN of Node 1 receives the incoming bundle and decapsulates it, delivering it to the NFD. The NFD of Node 1 confirms that the Data packet has a match at the PIT table (therefore it satisfies a pending Interest), saves the Data to the CS in case another user requests it in the near future, and delivers the Data to the relevant incoming face from the application that issued the Interest.



Data decapsulation and delivery to the application that issued the Interest at Node 1 (source node)

Extended Interest lifetime Another approach to extend the probability of receiving the requested Data in case of limited connectivity is to extend the lifetime of the Interest or re-issue the Interest after its expiration, thus creating a sense of permanent Interest. As a first step to achieve that, we experiment with the InterestLifetime field in the Interest packet and study the performance of the architecture in different scenarios. We note that longer lifetime causes more entries to be stored in the PIT tables, increasing memory consumption. Furthermore, in cases where the content provider is mobile, issuing long-lived Interests may lead to degraded performance, since fresh Interests will be suppressed before they have an opportunity to reach the content source. In cases, though, where the content provider is not mobile, the re-issuing of Interests, along with a suitable suppression mechanism, will provide an open route between the content provider and the end-users, minimising, at the same time, unnecessary retransmissions. This scheme can be employed by civil protection services that need to send emergency data to end-users.

4.1.2 Manual

Below we detail, the system requirements, the prerequisites and the installation guide for the integrated NDN – IBR-DTN platform.

System requirements

Ubuntu 14.04, 64bit

1GB RAM

Prerequisites

python >= 2.6

libsqlite3

libcrypto++

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

pkg-config

libpcap

Boost libraries >= 1.54

Installation guide ibrdtn

To install ibrdtn, download and run the ibrdtn-install.sh bash script using the following command in a terminal:

bash ./ibrdtn-install.sh

The script will download, unzip, configure, and install the necessary packages.

ndn-cxx_umobile libraries

Install the prerequisites entering the following commands in a terminal:

sudo apt-get install build-essential libcrypto++-dev libsqlite3-dev libboost-all-dev

sudo apt-get install doxygen graphviz python-sphinx python-pip

sudo pip install sphinxcontrib-doxylink sphinxcontrib-googleanalytics

Finally, build the ndn-cxx_umobile libraries entering the following commands in a terminal:

./waf configure

./waf

sudo ./waf install

ndn-dtn

Install the necessary prerequisites:

sudo apt-get install pkg-config

sudo apt-get install libpcap-dev

Finally build ndn-dtn running the following commands:

./waf configure

./waf

sudo ./waf install

Use the following command to create the proper configuration file:

sudo cp /usr/local/etc/ndn/nfd.conf.sample /usr/local/etc/ndn/nfd.conf

Running

Enter the following command to start the ibrdtn daemon:

dtnd

Enter the following command to start the nfd daemon:

nfd-start

Use the following command to check the status of the available interfaces, including DTN:

nfd-status

Documentation

To integrate the ibrdtn implementation with the rest of the UMobile platform, we implemented, among others, two new classes, DtnChannel and DtnFactory. The basic functions of these classes are provided below:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

The DtnChannel class implements the following functions:

listen()

accept()

handleAccept()

The DtnFactory class implements the following functions:

createChannel()

createFace()

getChannels()

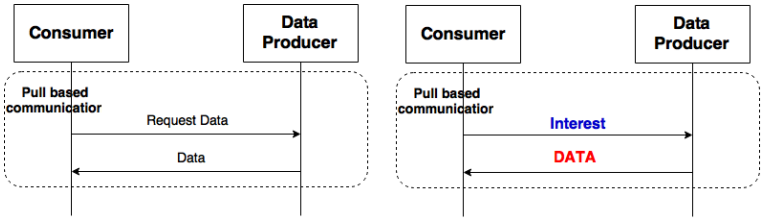
findChannel()

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

5 UMOBILE Services

5.1 Push Services

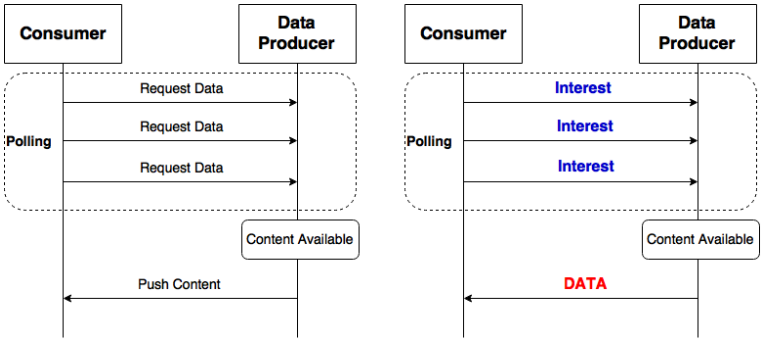
Based on the scenario requirements, UMOBILE platform requires both pull and push-based communication models as mentioned in D3.3. The pull-based model is inherently supported by NDN. Clients can request the desired content by issuing Interests with name prefix. Content Providers or any other NDN node with the corresponding content cached can respond forwarding data towards the Interface regarding information in the PIT. The Figure below illustrates the pull based communication model under NDN architecture using Interest/Data exchange.



Pull based communication model

By default, push based communication model is not inherently supported by NDN architecture. However, we can adopt several models of push-based communication by using default Interest/Data exchange of NDN architecture. The details of different push based communication models can be summarised as follows:

Push based communication model through Interest polling When the consumer does not know the exact time of data generation, it can keep polling the producer, issuing periodic Interests. Whenever the requested data is available, the data producer can push the content in a Data packet. In general, polling can induce network overhead and the freshness of the received data depends on the polling frequency. However, in some scenarios such as emergency situations where the consumer cannot associate with any UMOBILE base station, this model can be useful to retrieve the emergency services or content. The Figure below illustrates an Interest/Data exchange as a push based communication via polling.

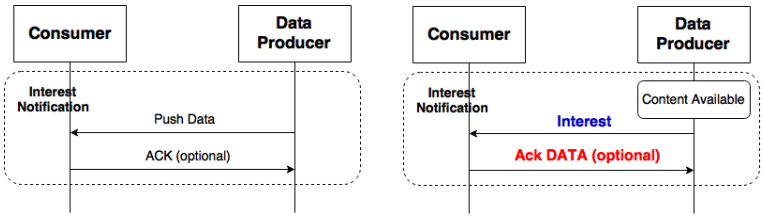


Push based communication model through Interest polling

Push based communication model through Interest notification In this model, we consider that the content's format is generated as text type (i.e. an instance message or a text notification sent by a server). Then, content

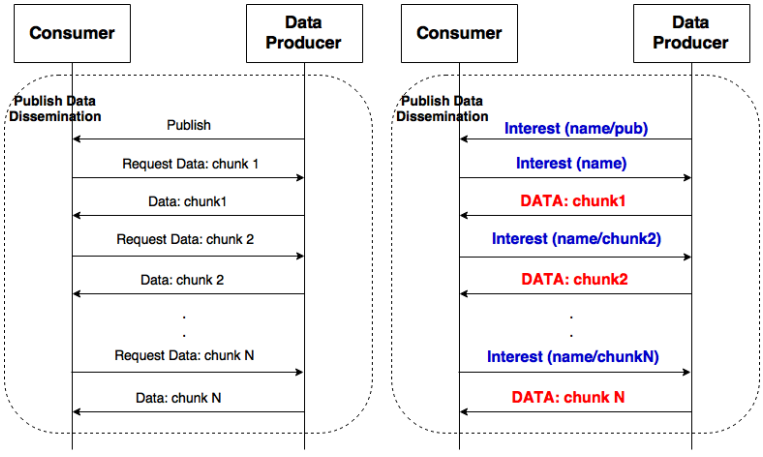
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

can be directly sent by appending it to the Interest name. This method is commonly referred as Interest notification. Upon the reception of such an Interest, the receiver can optionally send an ack Data to confirm Interest reception. The push based communication via Interest notification model is illustrated in the Figure below.



Push based communication model through Interest notification

Push based communication model with publish data dissemination In this model, we follow the publish – subscribe model where data producer can publish their contents or services via Interest message to the subscribed consumer which in turn trigger an Interest back from the consumer to fetch data. The Figure below illustrates the Interest/Data exchange of push based communication with publish data dissemination under NDN architecture. The data producer initially sends the publish message to its subscribed consumer. To distinguish Interest message with other models, new name component named "publish" is added after the content name. Consequently, the consumer receives the publish Interest message, it discards the last component ("publish") and send the new Interest message with the content name to request the data. In NDN, content is divided to several chunks. Due to self traffic regulation feature of NDN, N-1 subsequent requests are sent to retrieve all chunks of the requested content. The last component in name structure is reserved for the incremental chunk ID which will be automatically generated regarding previous received data chunk.



Push based communication model with publish data dissemination

Generally, in the NDN architecture, the Data message is expected to be sent after receiving an Interest message. However, in this model, the consumer responds to the first published Interest with new Interest message. In this context, we use NDN API to control the arrival of Interest messages by filtering names with key word "publish". If the Interest's name contains this keyword, the communication is operated through the publish data dissemination scheme.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

5.2 Network Contextualization Services: PerSense Mobile Light

5.2.1 Description

The introduction of new, cooperative technologies and in particular of low-cost wireless access, allowed the regular citizen to profit from the Internet as a commodity. This implies that most of the mobile devices (cellular, wireless) that have taken part on the Internet up until now as plain end-user devices, today can also be seen as networking nodes, having a role on the network operation. The movement that these devices have therefore impacts the underlying connectivity model and as consequence, it also impacts the network operation. Hence, being able to characterize such movement and also to estimate potential individual as well as aggregate movement is a requirement from the perspective of networking science evolution. This need goes beyond the integration of movement prediction and/or anticipation mechanisms in the network operation e.g., in routing or mobility management. In fact, roaming behavior is becoming more relevant, and today, due to an extensive effort derived from several initiatives as well as from extensive and wide traces collections, it is globally accepted that there is a relation between social behavior and the user's roaming behavior. It is the social behavior that assists in defining future user movement, both from an individual perspective, and from a group perspective. Hence, being capable of estimating such behavior is therefore 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 based on potential future moves of specific devices -; from a routing perspective – e.g. making routing more stable by selecting a priori paths that have a chance to be more stable in the presence of node movement; from a service migration perspective.

In the context of UMOBILE, PerSense Mobile Light is a service that has been developed to assist in performing network contextualization. Currently, PerSense Mobile Light captures information concerning a user's affinity network (contacts derived from Wi-Fi Direct and Bluetooth) as well as concerning roaming habits, over time and space (Wi-Fi). In a future version, PerSense Mobile Light shall collect data on user behavior, derived from additional sensors – the sociability forecasting module.

The current development phase (Phase I) was released in May as PerSense Mobile Light v1.0. This service allows data capture that assists in understanding preferences in terms of roaming habits and visited wireless networks. This type of contextualization is relevant, assuming, for instance, that a UMOBILE mobile device may present a roaming preferences model, where the intent is to consider personal preferences in terms of visited wireless networks. Or, it may contain a model that relates with the need to share data opportunistically based on frequency or volume of wireless contacts. These models shall be part of Phase II of PerSense Mobile Light (v2.0, December 2016), and shall assist a user definition, based on specific parameters. The identified user context is then used, together with the collected sensing data, to infer patterns of user behavior. This support is coined as "sociability forecasting" in the context of UMOBILE (refer to D3.3) and shall integrate aspects such as user recommendations and estimation of conditions for social interaction to occur derived, e.g., from shared interests; affinities; wishes.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

5.2.2 Tutorial

PerSense Mobile Light v1.0 can currently be used in Android devices. For that purpose, the researcher needs to join the Google Beta Community Pervasive Sensing Experimental Tools Community `https://plus.google.com/communities/104874036636715946374`. After this step, a URL will be available to download the tool. This first service version has as main purpose to assist the academic community in gathering meaningful traces and develop scientific studies, by reusing traces. The service captures wireless footprinting aspects such as geo-location; Access Points crossed and used; visit type and duration. It also captures affinity networks (devices around, via Wi-Fi Direct). This data is dumped daily to a server database (SQLite) provided by partner Senception. The traces are expected to be open via the UMOBILE Lab as well as via Crawdad, after adequate treatment (e.g. MAC obfuscation). The data is also stored on the device per day; per week, and for one month. During night, the data is dumped to the UMOBILE server. The data can be easily exported to usual formats such as cvs, or xls.

The database comprises several tables:

- Roaming tables:
  - 16 tables, 1 for each day of the week; 1 for each week 1 to 5; 1 for the current month.
  - Table entry tupple for each AP: *<Id, bssid, dayoftheweek, state, ssid, attractiveness1, lastgatewayIP1, dateTime, lat, long>*
- Visits table
  - Connected APs, entry tuple: *<Id, ssid, bssid, timeon, timeout, dayoftheweek, hour>*
- Affinity Network tables
  - 16 tables, 1 for each day of the week; 1 for each week 1 to 5; 1 for the month
  - Table entry tupple for each device in the vicinity: <Id, bssid, dayoftheweek, state, ssid, attractiveness$*$, lastgatewayIP$*$, dateTime, lat, long>

1 – fields to be worked in version 2.0 only.

The data is stored per device (1 directory per device, and per day). The SQLite database is stored as e.g. 19-04-2016-persenselight.db. This database can then be accessed with a command to edit SQL databases, such as the sqlitebrowser CLI, and the different tables can then be easily exported to a format such as cls or csv.

5.3 Communication Services: the Oi! app and the SOCIO framework

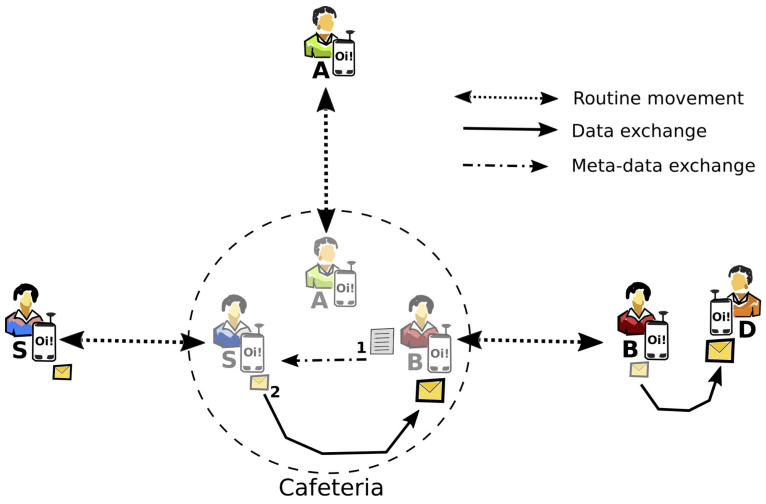
Despite all the technological evolution we have witnessed in the last years concerning devices' capabilities and networking paradigms, applications still rely either on the presence of wireless infrastructure or Internet access to allow data exchange among users.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

With Oi! [1] and SOCIO, we follow a social-based opportunistic networking paradigm to decouple the application from the dependency on the existence of Internet access: by exploiting the direct wireless communications capabilities (i.e., Bluetooth and Wi-Fi direct) available in personal mobile devices, SOCIO measures the social proximity among users, and allows Oi! to exchange messages independently of the availability of Internet access.

Oi! and SOCIO are expected to run over devices carried by users throughout their daily routines. Since these users socially engage in different settings (e.g., home, work, school) and with different others (e.g., family, friends, neighbours, colleagues, classmates), opportunistic data dissemination based on social awareness seems to be a rather interesting approach: Oi! messages are exchanged only between socially well-connected devices (i.e., users with strong social interaction). The reason for such choice refers to the fact that social similarity tends to vary less than node mobility (forwarding table updates are less frequently) and is dynamic enough to allow wise selection of next best forwarders (i.e., intermediate users) according to the social setting in which the involved parties find themselves [2][3][4][5][6].

The Figure below illustrates how Oi! messages are opportunistically exchanged through SOCIO considering the levels of social interaction among users in a working setting.



Social-aware opportunistic data transmission

As aforementioned, users meet one another in different settings and at different levels. In the Figure above, the source node S has a message for destination node D. During lunch time, node S meets node B that happens to work (i.e., spends more time in the same social setting) with node D. By considering the level of interaction of all nodes located in the cafeteria towards node D, node B is the best next forwarder to reach it given the social engagement in such social setting.

Based on that, Oi! messages are replicated only to socially relevant nodes (i.e., node B in this example) and refrain from using resources that will not result in the delivery of the message (node A could have received a copy of the message, but such effort would be useless as it does not socially engage with D at all in such setting).

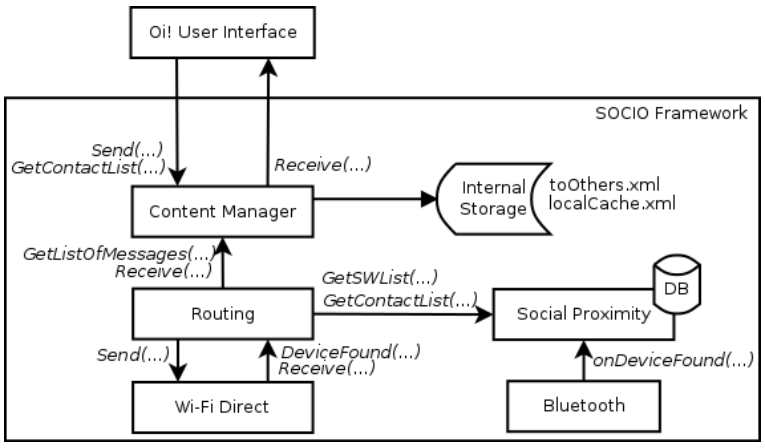
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

5.3.1 Oi! and SOCIO High Level Design

The Oi! application and SOCIO framework follow a modular design, aiming to allow easier extensions, such as the inclusion of other routing schemes, a novel algorithm to infer social proximity, or a new method to manage data. As illustrated in the Figure below, the Oi! application comprises the interface module, while the SOCIO framework comprises the remaining modules as explained next.

Oi!'s **User Interface** (UI) module allows the user to compose a message, choose its recipient from a list of known contacts, and see received messages. The UI interacts with SOCIO's **Content Manager** (CM) module, which is responsible to manage all the messages to be sent, as well as received messages. The primitives Send(destination, message), allows CM to build a message to be sent to a given destination node; Receive(message), allows CM to pass the received message to the UI; and GetContactsList(), fetches a list of known contacts to display on the UI. It is worth mentioning that a new application (e.g., dissemination app) will make use of these primitives provided by SOCIO's CM.

Based on the data provided by the UI, CM starts by composing a new message, which is stored together with the <sender, destination, timestamp> tuple. The sender and destination fields are the deviceID of the message's source and recipient devices, respectively. The timestamp is used to detect possible message duplication. This file is then stored in the Internal Storage (IS). All messages to be sent are stored in toOthers.xml file. Upon getting a new message from the Wi-Fi Direct through the primitive Receive(received message list), CM considers the tuple <sender, destination, timestamp> to filter messages. The filtering process is needed to check: i) whether the same message has been previously received; ii) if the message destination is the device itself, in which case checks if UI is available. If so, CM passes the received message to be shown in UI. Otherwise, CM saves the message to the localCache.xml file so it can be delivered when UI is available.



Oi! and SOCIO high level design

Routing (RT) module implements dLife, a social-aware opportunistic algorithm based on daily routines [3]. Forwarding decisions are based on the level of social interaction among users: message replication only takes place when the social weight of an encountered node towards a specific destination is higher than the social weight of the current carrier of the message towards such destination. When a device is found, through the primitive DeviceFound(deviceID, SW list) triggered by the Wi-Fi Direct module, RT gets the identification and list of social weights of

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

the device and by using the primitives GetListOfMessages(list of destinations) and GetSWList(list of destinations), RT gets a list of carried messages and the social weights towards a specific list of destinations, respectively, to help with forwarding decisions. When the forwarding decision is made, RT uses Send(destination, message list) to provide the Wi-Fi direct module with a list of messages to be replicated to the destination (i.e., encountered Oi! device).

Wi-Fi direct module comprises the notification and data exchange phases. In the notification phase, this module detects the presence of other Oi! devices and notifies RT. As for the data exchange phase, this module i) considers the list of messages from RT to replicate to an encountered Oi! device; and ii) provides CM with a set of messages received from another Oi! device.

Social Proximity (SP) module computes the social weight towards encountered devices through Bluetooth sightings and stores this information on its database. This social weight is given by the Time-Evolving Contact Duration (TECD) [7], which via the primitive onDeviceFound(deviceID, encounter time) compares the encounter time against the current time after a Bluetooth scan to determine the contact duration towards a specific device (i.e., deviceID). Social weight is computed in a hourly base, which can include different contacts of various durations. SP provides the UI with a list of known contacts through CM, and RT with a list containing the potential destination devices and the social weight towards them.

Bluetooth (BT) module is used to detect the presence of neighbouring devices. BT provides SP with the identification of the neighbour device and the time of encounter.

5.3.2 Manual

For a full description of Oi! and SOCIO, including the flowchart operation, future work, and code documentation, refer to Annex 1. Respective codes are available at:

- GitHub Oi! – `https://github.com/COPELABS-SITI/Oi`
- GitHub SOCIO: `https://github.com/COPELABS-SITI/SOCIO`

5.4 Service migration

5.4.1 Description

The objective of service migration is to facilitate efficient and resilient service delivery from network operators/service providers to network edges in order to support the necessary Service Level Agreements (SLA) as well as accommodate service reachability in challenged networks. As mentioned in D2.1, UMOBILE needs to support various challenged scenarios such as the aftermath of disasters or rural/remote network deployments or networks with limited backhaul capacity. Such challenged environments poses several challenges including increased latency, intermittent connectivity etc. To address these challenges, we propose a resilient service migration platform that utilizes advances in lightweight operating systems such as Docker containers to push service instances right to the network edge. Inside a local network, service migration module utilizes a name based routing/forwarding strategy

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

providing the benefits of ICN. Within the UMOBILE architecture, we will utilise the NDN abstractions to carry out the name based routing/forwarding strategies.

5.4.2 Manual

The fundamental building blocks of service migration includes the following five main modules:

Service Execution Gateway (SEG): This is the point of attachment for clients which is usually a wireless access point deployed in UMOBILE network. A SEG provides a virtualized environment to store and execute services locally upon a user request at the edge. The SEGs also carry out the necessary translations between IP and NDN providing support to current IP based services, as well as removing the need for changes to the end user equipment.

Service Controller (SC): SC manages the mapping of publishers of services and subscribers of services. The SC offers the repository to store services and delivers the service image over the network. It regularly disseminates the list of available services including the updates from the service publisher to all nodes residing in the network. It also collects information from the network such as devices capabilities, topology information, service usage patterns etc. This information will be then used by the SC to make optimal service migration decisions.

Service Publisher (SP): SP refers to the original content producer. SPs register services by publishing the services to the SC.

Gateway (GW): This is responsible for connecting different domains (two separate networks). GWs also carry out the necessary translations between the domains (e.g. NDN to IP and vice versa).

Forwarding Node (FN): This is responsible for routing requests for services towards available copies. These nodes also cache services locally, thereby allowing local copies to be returned to the client (they do not execute services though).

The operations in service migration can be explained as following:

Registration To migrate services in UMOBILE network, the service providers must register their service to the service controller. As service migration relies on Docker container, the service providers could proactively prepare their services as image files (e.g., docker image, tar file) and upload them to the main repository inside the service controller. This operation is carried out through IP or NDN. In case of NDN, the service provider uses the push based communication model with publish data dissemination to push the service to the service controller. In this operation, the service provider acts as a data producer, while the service controller is considered as a consumer. The service provider initially sends the Interest message towards the service controller. An example of the Interest's name can be illustrated as /umobile/register/<service-name>/<version>. Accordingly, the service controller sends another Interest message back to the service provider to retrieve the image file and store in the main repository.

Discovery Each SEG must be aware of the available services in their network. A SEG has to subscribe to the SC to retrieve the latest update of available services. This information is responsible for a list of services that are locally available in the network. This operation can be done through the Interest notification communication model. SC acts as the data producer and appends the service name to the Interest message's name. As a consequence,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

the SEG (consumer) extracts the service name and updates its local table. An example of an Interest's name can be illustrated as /umobile/discovery/<service-name>/<version>.

Monitoring All registered SEGs have to periodically monitor the service usage indicated in the list of available services, as well as the node usage and network condition. This information is retrieved by the SC in order to decide when the service should be migrated and where the service could be operated. In this operation, the pull based model is considered while the SC and SEG operate as consumer and producer respectively. An example of an Interest's name can be illustrated as /umobile/monitoring/<SEG-ID>/data.

Migrating SC must be able to utilise all information from the service monitoring and decide to push the required service to particular SEG. In this operation, the push based communication model with publish data dissemination is considered, since the content is non text type (e.g., tar file) and the communication is initiated by the data producer (SC). An example of Interest's name can be illustrated as /umobile/migration/<SEG-ID>/<service-name>. SC specifies the target SEG (consumer) through <SEG-ID> component and appends the service name to the last component. When the target SEG receives the Interest message, it can reply back to the SC with a subscribed Interest message. Then, the service image can be sent from the SC to the target SEG as mentioned in push based communication model with publish data dissemination section. Note that the service image can be sent from any intermediate forwarding nodes who have the requested service in their cache.

5.5 Smart routing

UMOBILE's smart routing is expected to provide support for urban, remote, and disaster areas where users have specific applications and (maybe) infrastructure to allow communication among them, and such communication may follow a host-based (i.e., towards a specific host) or content-based (i.e., towards different interested parties) approach.

In regards to the aforementioned networking areas, the most common is the urban area, which is well served of infrastructure (e.g., Wi-Fi , cellular, and cable/fiber networks) and Internet connectivity. This area is characterised by being rather dense, comprising many fixed and mobile networking-enabled devices.

As for the rural area, it has scarce infrastructure and limited Internet connectivity. This area is comprised by a small collection of fixed nodes spanning a great geographical area with few mobile nodes moving between such node agglomerations to allow communication.

Finally, the disaster areas (which can include both urban and rural settings), which rely on no infrastructure (as they may have been damaged by an earthquake, for instance) and takes advantage of whichever communication technology is available, which makes it rather disruptive.

Concerning the communication models, the pull-based (i.e., content-based) or the push-based (i.e., host-based) approaches can be used in any of these areas according to the target application (e.g., a user enjoying a chat application would resort to a host-based communication paradigm to reach a specific peer; while a user interested in receiving concert notifications would probably use a content-based application that receives content matching

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

such user's interest). Both communication models are orthogonal to these areas with respect to the applications operating over them.

These different areas have been subject of research for many years now, and each of these networking domains have very specific challenges (e.g., at an urban area routing must cope with interference and multiple forwarding paths; at a remote area routing must deal with intermittent connectivity; at a disaster area routing must overcome the lack of readily available infrastructure). The same applies to the communication models, which have been exploited across these networking domains for quite a while now, but normally considering very specific target applications.

Still, there is no smart solution that is able to operate across these different networking areas employing both the aforementioned communication models. Thus, in UMOBILE we introduce a smart routing framework capable of operating over different scenarios regardless of their inherent networking challenges and desired applications.

For that, this smart routing solution must:

- Consider usage, network, and user context.
  - The usage context refers to the individual aspects of the use of the system (e.g., individual preferences). Hence, the usage context is captured in a non-intrusive way via the mobile device, e.g., by locally analysing aspects such as logs of the different applications a user relies on, social networking logging (e.g., likes, dislikes), browsing history, among others.
  - The network context can be retrieved by tracking the user networking conditions. It can be used to adapt routing to the area of action (e.g., crowded urban area or sparse network) and the condition of each network interface (connected/disconnected, packet loss, etc). This could also differentiate the mode of operation between connected and disconnected mode, for instance. This information can be used to decide what is the best network interface for sending a certain packet.
  - The user context is based on physical and social information. User physical information refers to the surroundings of the user, number of nearby peers, his movement patterns, location, among other physical-related aspects. Concerning the social information, it relates to levels of social interaction, trust circles, shared interests, and others aspects that bring users together and that concerns propinquity (i.e., social nearness) and proxemics (i.e., human social space). This set of information allows us to take advantage of applying inference of user context to better deliver data. The more we know about the users, the more improved data exchange among them: context information facilitates decision making in what concerns routing solution. By taking into account where users are, how they move, with whom they interact, how long they spend with others, and their preferences are few examples of user context that have been successfully employed in routing, even though only from the physical perspective (e.g., contact time) [2][3][8][9][10][11][12][13]. It is also relevant to consider aspects that are related to social nearness, as well as to proxemics and its impact in social interaction and consequently in data transmission.
- Take advantage of all of the feasible contact opportunities among users' devices. Users are highly mobile, and such mobility is rather challenging for routing (short-lived contacts, multiple data exchange paths, high number

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

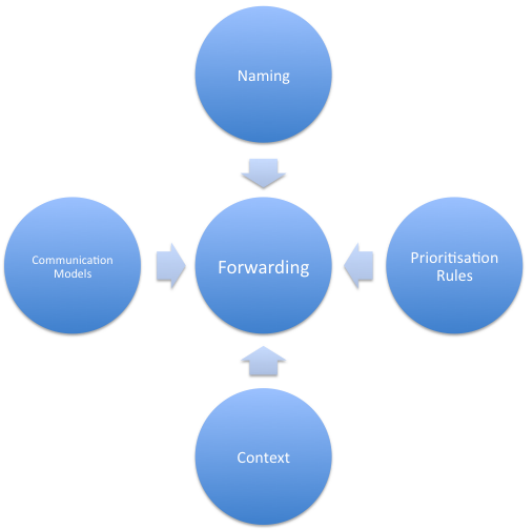
of contacts). Thus, routing must make the most out of such contact opportunities, focusing on those which are indeed relevant and shall maximize content delivery;

- Provide support to pull- and push-based communication models. At the same time users want to receive content of their interest, they also want to be able to reach a specific peer (e.g., VoIP call). This can be achieved by employing pull- and push-based models, in which communication may be initiated by the interested party and content source, respectively. For that, routing should take into account the different types of user applications to allow such communication models;

- Employ an application-driven prioritisation mechanism (emergency messages have priority over emails, for instance). The type of application should also influence the order that messages are disseminated. A message about a wild fire must have higher precedence over a simple chat about a soccer match. This routing feature is desired as it allows better user experience, especially in situations that may put in cause human life;

- Be compliant with different networking paradigms (ICN, DTN, OppNet, IP). Irrespective of networking domain in which the users find themselves, a smart routing approach must be able to allow the exchange of information that is seamless to users. A user shall be able to send and receive information, with smart routing deciding which networking paradigm flavour to employ as to allow for improved delivery capability with low associated cost and latency.

These relevant aspects are brought together to allow for a smart routing framework that can cope with the demands of the aforementioned networking areas. The Figure below illustrates the proposed routing framework and its main components, namely context, naming (cf., Section 5.6 for further details), prioritisation rules, communication models, and forwarding.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

5.5.1 Context

The UMOBILE platform assesses context based on i) application usage, ii) network state, and iii) user-related information. Context as a whole shall have an important role in determining the importance of each data piece, the employed naming scheme, the forwarding approach to be used, and the communication model that best suits the current user needs.

Usage context In order to differentiate the network requirement for each one of the UMOBILE applications, we need to retrieve information from the application data. To this end, the usage context module is responsible for retrieving this information from the application data using metadata included in data packets. This information must be used to adapt the forwarding options to the requirements of each service (e.g., providing more priority to some services). Moreover, services are often faced with stringent requirements in terms of performance, delay, and service uptime. On the other hand, little is known about the performance of applications in the network, for instance the response time variation that is induced by network connectivity and traffic load. The usage context information adapts forwarding to service requirements imposed by applications, potentially improving the content and service delivery.

Network context There are three major scenarios that will allow for network contextualisation.

- Wi-Fi hotspots with/without internet connectivity: a networking scenario where a set of users are connected to a Wi-Fi hotspot that centralizes the communications inside a Basic Service Set (BSS). Each Wi-Fi hotspot can be connected to the Internet, or it can be connected to another Wi-Fi hotspot providing interconnectivity between different Wi-Fi areas.
- Opportunistic networks based on Wi-Fi-direct: a networking scenario where users must deal with opportunistic communication. That is, users do not have pervasive connection to the Internet, and they can have temporary encounters with other users, creating temporary connections between them.
- Poorly connected/isolated areas: a networking scenario where a set of users are isolated from the network, but a mobile device interconnects them with the rest of the network using DTN capabilities. This mobile device acts as a data mule, and it can be a UAV device or any other vehicle, such as a public vehicle. Delay-tolerant forwarding can be beneficial in cases where data needs to be forwarded over vehicles, such as UAVs or emergency vehicles. This kind of connectivity can be deployed to provide communications in remote areas or in emergency cases.

User context User context is captured based on local physical and social information. Such information is to be acquired by means of a non-intrusive service which resides on the end-user device. By non-intrusive it is meant that this service takes advantage of the natural networking footprint that is overheard by devices, be it via WiFi and Bluetooth, as well as any other means (e.g., LTE Direct). In UMOBILE, however, the effort will be focused on short-range wireless in the form of WiFi and WiFi Direct.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

User context can be defined by the following set of information:

- Physical information: derived from wireless connectivity, it incorporates aspects such as duration of visits, common local places and paths, movement pattern, location, nearby peers, among others.
- Social information: trust circles, shared affinities, wishes, or interests, levels of social interaction (e.g., family, friends, familiar strangers), and any other social information that may be derived from the notions of propinquity (social nearness) and proxemics (human social space).

5.5.2 Prioritisation Rules

Prioritisation rules take into account sociability forecasting. Sociability forecasting concerns predictive analysis of interaction to occur derived from shared interests, affinities, i.e., derived from a correlation between individual user context. Such user context is captured based on local connectivity as well as device usage, on the end-user device. It may or may not be applied together with networking policies. Nonetheless the most relevant aspect is that sociability forecasting shall consider proximity of devices to improve routing aspects such as successor selection.

Sociability forecasting concerns user and usage context, .e.g., sensing data about users' roaming and relative location, by using the Wi-Fi interface; about users' social interactions, by using the Wi-Fi and Bluetooth interfaces, and about users' behaviour, e.g., by using data concerning device usage. Such "smart" captured data (in contrast to raw data) is then made available to a contextualisation and behaviour inference module as well as to the routing/data sharing module. As the data resides solely on end-user devices, there is no endangering of data privacy.

The identified user's context is then used, together with the collected sensing data, to infer patterns of user behaviour thus assisting in developing priority rules dynamically, e.g., by taking into account aspects such as crowd density. The inference process may be distributed among personal devices or may also include cloud computing entities, depending on the amount of data to be analysed, the required learning algorithms, as well as the delay tolerance and privacy levels of applications. Similar distribution of computational effort may be needed to adjust contextualisation modelling, by considering quantifiable social parameters, and by adjusting them to the roaming dynamics that can characterise user's behaviour with an adequate level of assurance. Some aspects that are considered in the notion of user context are:

- Social trust circles. These correspond to networks of devices owned and carried by users that share affinities, wishes, or interests. Trust circle computation assist in collective inference, derived not only from physical proximity, as well as from social proximity. Examples of trust circles are groups of friends; familiar strangers interested in a specific event.
- Social roaming footprint. The footprint that both individual and collective users exhibit when roaming around. Derived from wireless connectivity, it incorporates aspects such as duration of visits; common local places and paths.
- Personalised recommendations. Recommendations are filtered by the devices based on the personal and individual affinities of each user.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

5.5.3 Communication Models

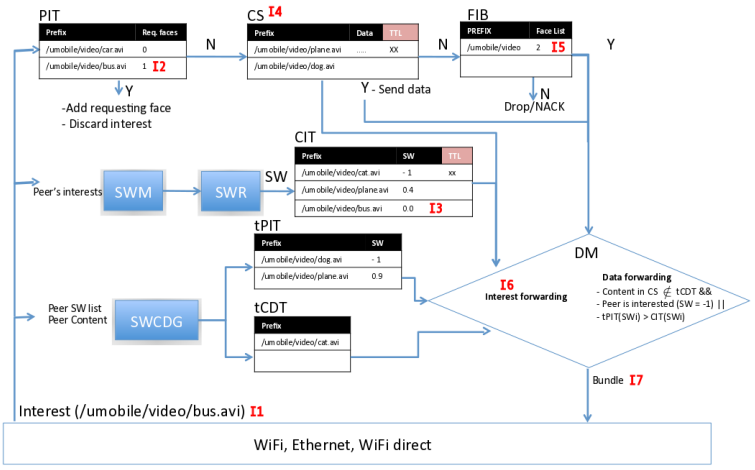
Whether it is pull- or push-based, the employed communication model considers the type of application running over this smart networking framework. Smart routing can be beneficial to both pull- and push-based UMobile applications. Indeed, the path between a content consumer and content producer is constructed in a hop-by-hop fashion as the Interest packet traverses the network, so employing a smart routing protocol to forward Interest packets positively affects not only the upstream path (employed by push-based applications) but also the downstream path (employed by pull-based applications).

5.5.4 Dynamic Forwarding Engine

An engine that provides support to operation over different networking domains and according to the user context and prioritisation rules. This forwarding engine also considers the naming scheme set by the naming component, and follows the communication model defined by the user application.

As mentioned earlier, the proposed smart routing framework is expected to provide seamless data exchange among user devices, irrespective of the area the user may be traversing at a giving moment. For that the forwarding engine encompasses a set of data structures, comprises three distinct phases (interest tracking, interest forwarding, and content passing phases), and will rely on the remaining components as to take the most suitable decision. It is worth mentioning that UMOBILE integrates NDN and DTN to allow for cross-networking domain operation. Thus, interest packets are bundled and exchanged opportunistically until a node holding the content responds to that. Similarly, a data packet is bundled and follows the PIT breadcrumbs towards the interested parties.

The Figure below illustrates the proposed forwarding engine, being further detailed next.



Forwarding engine

Data Structures The set of data structures considered by the proposed forwarding engine may be persistent and temporal. Moreover, each entry on data and interest tables (cf., CS and CIT, respectively in the Figure above) has a

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

time-to-live (TTL), where i) data TTL refers to the usefulness of the content; and ii) interest TTL expresses the time in which a node is interested in a given content.

Persistent structures are:

- Content Store (CS) – responsible for holding information concerning the content carried by the current node. The content stored here allows for the implementation of the Store-Carry-Forward paradigm seen on DTN, in which the current node shall carry content to those with whom it is socially well connected and that are interested on it. It is the same as defined by NDN [14], with a slight difference: it now has a TTL field that allows for stale data removal.

- Carried Interest Table (CIT, similar to the PIT on NDN [14]) – responsible for keeping up-to-date information concerning the data interests of the current node along with its social weights towards other nodes with whom it socially interacts. This table also holds the interests of the current node (with SW set to -1) and contains only relevant social information for taking forwarding decisions. Note that the interests of the current node can be specified explicitly by the user, or generated by any of the guest applications enjoyed by the user. The difference towards the NDN PIT is that the stored Interests are kept even after being "satisfied", due to the replication strategy used by opportunistic routing. Interests are deleted based on TTL, when applicable.

Due to the intermittent nature of the connectivity wireless links, two temporary structures are used while neighbouring nodes have wireless contact. It is worth mentioning that the temporary tables must have their entries erased and replaced upon a new encounter or disconnection.

- temporary Pending Interest Table (tPIT) – holds the list of interests seen by the encountered node and the social weights between itself and such interests. The list also holds the interests of the encountered node (with SW also set to -1).

- temporary Carried Data Table (tCDT) – holds information about the data that encountered node is currently carrying.

Note that PIT and FIB as per NDN definition [14] remain the same to allow regular NDN operation.

Interest Tracking Phase The operation of the forwarding engine depends on the knowledge about the interest of neighbouring peers. This phase gathers information concerning the interests of the peers that socially engage to compute social weights.

The forwarding engine encompasses three modules in order to compute social weights towards a specific set of interests:

- Social Weight Measurer (SWM) – responsible for keeping track of the contact duration between the current and encountered nodes. SWM maps this duration to the encountered peers' interests. Based on that, this component shall compute the level of social interaction of the current node towards the interests that encountered peers have. This social weight determines how good the current node is to reach others with such identified

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

interests. Social weight computation takes place at every hour as to allow for a better view of the dynamic social behaviour of users in a timely manner.

- Social Weight Repository (SWR) – responsible for storing the list of interests the current node comes across (obtained upon encountering a peer). SWR holds the interest, time it has been first encountered, and the cumulative duration of within a specific hour. Upon a new hour, contact duration is updated and the social weight towards each encountered interests is computed for the last hour. In the case, the current node still 'sees' such interest (i.e., respective peer node is still in the vicinity), time of first encounter is updated and contact duration is accounted by SWM for the new hour.

- Social Weight and Carried Data Gatherer (SWCDG) – responsible for obtaining the list of interests and social weight towards them from other peers (peer SW list). This element is also responsible for obtaining information concerning the content carried by encountered peer (Peer Content). This information is used to help making forwarding decisions.

Social weight (SW) computation is triggered when a specific face notifies SWM of the presence of a neighbour node. SMW creates entries for this encountered node in SWR considering the interest information obtained by SWDCG. SWM keeps track of the contact duration between current and encountered nodes and updates SWR accordingly. Since nodes (i.e., their users) present different patterns of behaviour in different time periods, SW computation takes place in an hourly fashion. Then, SW information is written to SWR and updated to CIT. Information obtained by SWCDG from the encountered node is used to populate tPIT (with a view of its CIT) and tCDT (with a view of its CS).

Interest Forwarding Phase In this phase the Decision Maker (DM) may employ either a pull-based (i.e., initiated by the interested party) or a push-based (i.e., initiated by the content source) model as specified by the communication models component.

DM relies on the Content Store, PIT and FIB, and Interest and Data packets follow the regular NDN breadcrumbing approach with a subtle difference: now Interests packets can be bundled to allow for an opportunistic propagation throughout the system towards potential Data sources leaving a trail. When the bundle reaches a source with the desired content, the respective Data packet is bundled and follows the trail back to the content requester.

As per definition, NDN does not employ the push-based model: the content source (i.e., a caller) is not able to initiate a call as NDN is not based on hosts (i.e., the callee's address is unknown to caller). However, NDN can support the push-based model, in which the interested party shall manifest his/her interests prior to the existence of the content (i.e., the call), or extra signalling may be used by the content source (i.e., caller) to find the callee.

As a simple example, consider that the current node receives the following Interest packet: "/umobile/video/bus.avi" (cf., step I1 in Figure above). As it does not have a respective entry on its PIT, it adds such entry (step I2), and also adds such interest to its CIT (step I3, the node keeps track of this information to allow content-based social-aware data exchange as described next in Content Passing Phase). Then, the current node checks whether it has the content on its CS (step I4). Since data is not available, it resorts to its FIB (step I5) where it learns the content can

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

be reached by means opportunistic contacts. It relays the Interest packet to CM (step I6), which in turns bundle and place in in the outgoing queue for dissemination (step I7).

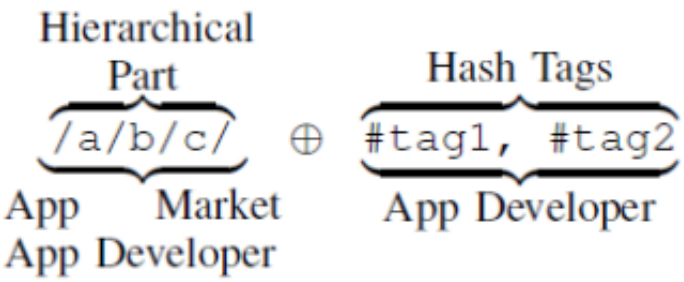
Content Passing Phase DM relies not only on the Content Store, but also on the CIT, tPIT, and tCDT data structures. This allows forwarding of data packets to take place at every hop (i.e., bundles exchanged upon an contact opportunity) and to be based on the level of social interaction towards specific interests. For this, DM interacts with CIT and tPIT to obtain social information of the current and encountered nodes, respectively. For every content in the current node's CS, forwarding shall only take place if:

- The content that is in CS is not in tCDT; AND
- The encountered node is interested on it (in this case "/umobile/video/dog.avi" is content to be relayed); OR
- The encountered node has higher social weight (tPIT) than the current node (CIT) concerning their interaction with others interested on the content to be replicated (since peer's SW = 0.9 > current node's SW = 0.4, "/umobile/video/plane.avi" is content to be relayed).

It is worth noting that DM should also take into consideration of the input from the prioritisation and naming components. Moreover, bundling will only occur if the requesting face does not refer to a guest applications running on the user device. In such cases, the content is sent directly to the application (i.e., its requesting face).

5.6 Naming

UMOBILE naming wil be based on the keyword-based approach described in D3.3 where the naming scheme is divided in two parts, a fixed hierarchical part and a set of hashtags used to define attributes.



Proposed naming scheme

The fixed hierarchical part follows the hierarchical naming scheme of NDN and its purpose is to guarantee compatibility between instances of the same or different services/applications. It is defined by application developers and they can define their own hierarchical name spaces, enabling the communication between different instances of the applications. Hashtags comprises of hashtag-like keywords, which the application developer can add to the application. The exact semantics of the hashtags depending on whether the fixed hierarchical part of the name corresponds to static content or an application function(ality). In the former case, these keywords are used to

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

semantically annotate the static content. This feature enables the partial matching of requests with available cache or routing/forwarding entries i.e., given an exact match in the fixed hierarchical part of the name, hashtags can be used to support approximate matching enabling the search of information in other devices. When the fixed part of the name identifies a certain application function(ality), the hashtag part of the name enables the passing of adequate parameters.

Since we follow NDN architecture as the base/starting point for the UMOBILE project implementation, the packet format will be based on NDN packet. In the following we describe this packet format and how it is used to implement our UMOBILE keyboard-based naming scheme aimed at improving NDN scalability on Mobile/DTN networks.

In order to represent UMOBILE naming scheme, we need to take into account NDN packet format and how NDN represents interests and data packets and the information included. In NDN, each packet is encoded in a Type-Length-Value (TLV) format. NDN Interest and Data packets are distinguished by the type value in the first and outmost TLV. An NDN packet is mainly a collection of TLVs inside the main TLV. Some TLVs may contain sub-TLVs, and each sub-TLV may also be further nested. A guiding design principle is to keep the order of TLVs deterministic, and keep the level of nesting as small as possible to minimize both processing overhead and chances for errors. Note that NDN packet format does not have a fixed packet header nor does it encode a protocol version number. Instead the design uses the TLV format to provide the flexibility of adding new types and phasing out old types as the protocol evolves over time. The absence of a fixed header makes it possible to support packets of very small sizes efficiently, without the header overhead. There is also no packet fragmentation support at network level. However NDN defines a set of TLVs as following.

Interest packets Interest packets are issued by clients in order to request some content. To respresent interests packets, NDN uses the TLV definition as follows. Name and Nonce are the only two two required elements in an Interest packet.

- Name: Hiearchical name for NDN content, which contains a sequence of name components. We should add to the name the hashtags used in the keyword-based naming scheme. This tags can include contextual information defined in D3.3, such as application, network or social context.
- Selectors: These are some optional parameters used to help interest matching with NDN data packets, such as min/max suffix components or if data must be fresh. For UMOBILE naming, we could add some selectors, for instance if we support partial matching.
- Nonce: A randomly-generated 4-octet long byte-string, which combinated with the name uniquely identify an Interest packet, used to detect looping Interests.
- InterestLifeTime: Indicates the time remaining before the Interest times out. This should be used in order to determine the DTN aware interests.

Data packets The Data packet represents the content elements together with its Name, some additional bits of information (MetaInfo), and a digital Signature of the other three elements. NDN Data packet is TLV defined as

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

follows:

- Name: Hiearchical name for NDN content, which contains a sequence of name components. We should add to the name the hashtags used in the keyword-based naming scheme.
- MetaInfo: This field is optional and can include the content type, the freshness period or the final block id. This optional freshness period FreshnessPeriod indicates how long a node should wait after the arrival of this data before marking it as stale. Stale data is still valid, but it can be filtered depending on the interests. This field can be used to define the data time to live in DTN connections.
- Content: This field is the content itself.
- Signature: NDN Signature is defined as two consecutive TLV blocks: SignatureInfo and SignatureValue.

Naming components The name in NDN is specified by name components and special markers. Both are defined below. We specify a third parameter called hashtag attributes, that will be the specific components to describe the different keywords used in the UMOBILE naming.

- Hierarchical name component: A set of components define the NDN hierachical name, e.g. /umobile/emergency/video. Each word is a different component.
- Special marker: NDN defines special markers in order to differentiate versions, timestamps and sequence numbers included in the name.
- Hashtag attributes components: In UMOBILE we define a third component in order to represent the keywords used in the UMOBILE keyword-based naming scheme. e.g. /umobile/emergency/video #fire #areax

Naming methods In the NDNx implementations a set of methods are defined in order to NDN naming defined methods (the main ones):

- name(): Create a new Name.
- appendComponent (): Append a new component, copying from value of length valueLength
- clear(): Clear all the components.
- appendVersion(): Append the version protocol to the name
- appendTimestamp(): Append a timestamp using the UNIX time.
- appendSequenceNumber(): Append a sequence number to differentitate the multiple chunks.
- getSuccessor(): Get the successor of the NDN hierarchical name.
- match(): Check if the N components of this name are the same as the first N components of the given name.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

- size(): Returns the number of components.

In order to support partial matching and taking benefit from UMOBILE keyword-based naming, we need to define a set of UMOBILE specific naming methods:

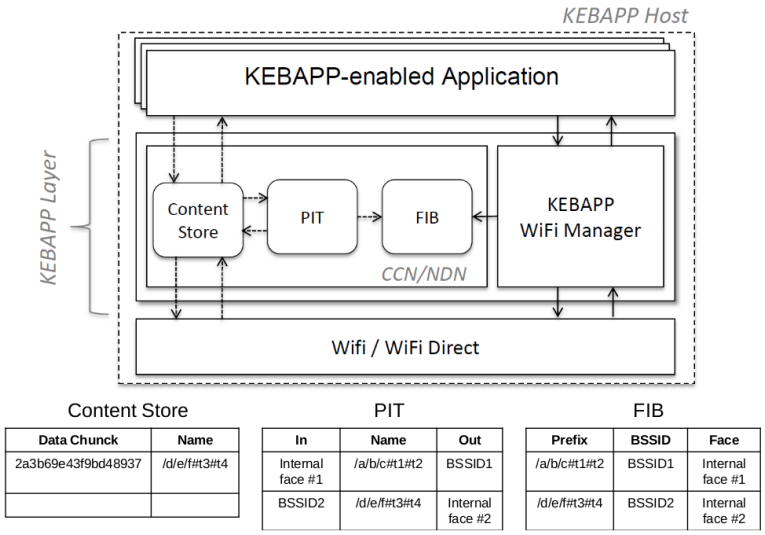
- fullMatch(): This full match extends the match method defined in NDN by comparing also all the different hashtag attributes used in the UMOBILE naming included in interest packet
- partialMatch(): This partial match extends the match method defined in NDN by comparing also the different hashtag attributes used in the UMOBILE naming included in interest packet, but it can ingore some of them.
- hasAttributes(): Check if the name contains hashtag attributes.
- getAttribute(): Return a certain hashtag attribute contained in the name.
- setAttribute(): Add a hashtag attribute component to the name.

5.7 Keyword-Based Mobile Application Sharing (KEBAPP)

5.7.1 Description

Within the UMOBILE project, we present KEBAPP, a new application-centric information sharing framework oriented to support and provide opportunistic computing to mobile devices (smartphones, tablets, etc.). Our approach targets scenarios where large numbers of mobile devices are co-located presenting the opportunity for localised collective information exchange, decoupled from Internet-access. In KEBAPP, we propose the creation and use of 802.11 broadcast domains for the support of particular applications i.e., KEBAPP-enabled hosts or APs advertise one or more Basic Service Set(s) (BSSs) for the support of one or more application(s). The creation of application-specific BSSs aims at enabling mobile devices to connect only when their counterparts support the same application and/or namespace. The advertising AP or host, through a WiFi Direct Group, acts as a mediator to connect different users willing to share the same application in a single broadcast domain. In the case of APs, functionalities such as access control, association, encryption, etc., can be supported without imposing computation and/or battery overheads to mobile devices. In this context, KEBAPP employs application-centrism to facilitate/enable (i) the exchange of processed information, in contrast to merely static content, and (ii) the discovery and delivery of information partially matching user interests. The Figure below presents the structure of a KEBAPP-enabled host. KEBAPP provides a new layer between the application and the link layers exhibiting three major design features. Namely, (i) application-centric naming, where applications share common name-spaces and further support the use of a keywords, (ii) application-centric connectivity management (KEBAPP WiFi Manager), where applications manage connectivity by defining and/or joining WiFi broadcast domains, and (iii) information-centric forwarding, extending NDN architecture.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124



Content Store

Data Chunck	Name
2a3b69e43f9bd48937	/d/e/f#t3#t4

PIT

In	Name	Out
Internal face #1	/a/b/c#t1#t2	BSSID1
BSSID2	/d/e/f#t3#t4	Internal face #2

FIB

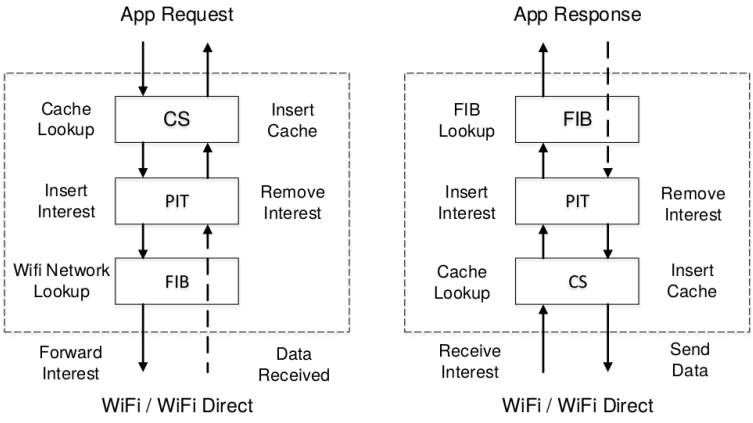
Prefix	BSSID	Face
/a/b/c#t1#t2	BSSID1	Internal face #1
/d/e/f#t3#t4	BSSID2	Internal face #2

KEBAPP-enabled host

KEBAPP Operation The basic forwarding operation of a KEBAPP node is a modified version of NDN architecture, aimed at reflecting the forwarding of messages within the various Basic Service Sets (BSSs) a node may participate. As explained in the following, since we consider single-hop broadcasting domains, forwarding decisions lead to either the broadcasting of a message in the BSS or its delivery to a local application instance. As such, broadcast domains are considered as (inter)faces of a KEBAPP node.

The KEBAPP forwarding scheme, similarly to NDN, has three main data structures: FIB (Forwarding Information Base), CS (Content Store) and PIT (Pending Interest Table). The FIB is used to forward Interest packets toward potential sources of matching data. The CS is responsible of caching the information requested by the users, providing fast fetching for popular information and avoiding to recompute information already requested by other users. The PIT keeps track of Interests forwarded to any BSS.

The Figure below provides a representation of the KEBAPP packet forwarding engine.



Forwarding operations

Below, we detail the operation of KEBAPP framework for an information requester:

1. The application requesting for information creates a new Interest.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

2. The application looks for the information in the local CS. If the information exists locally, the data is sent to the application.

3. If the information is not found in the CS, the KEBAPP layer inserts an entry in the PIT (<Internal_Face, name_prefix + keyword_list, null>). As in NDN, we use the term “Internal Face” to point to the local application involved in the transaction (either as requester or as provider).

4. The KEBAPP (network) layer checks if there is a BSSID entry in the FIB matching the name_prefix of the PIT. 5 If an entry for the requested name prefix exists, the WiFi manager con- nects the WiFi interface to the BSSID in the FIB and broadcasts the Interest message with a corresponding time-out value.

5. Each time a new FIB entry is added because a new prefix name is discovered on a new BSS (e.g., through WiFi NAN), the KEBAPP layer checks if a pending PIT entry for this prefix exists. As mentioned above, this corresponds to PIT entries created for Interest messages that could not be forwarded. In case an entry exists, the Interest is sent through the recently added BSSID, and the entry is updated with the BSSID value.

6. When a response is received with the information requested, the KEBAPP layer looks for the internal face that points to the application in the corresponding PIT entry and fordwards the response to it. The PIT entry is removed and the information requested is cached in the CS.

Next, we describe the operation of the KEBAPP framework for an information provider:

1. The user receives an Interest through the interface connected to a certain BSS related to an application.

2. The KEBAPP layer checks the CS for a matching entry.

3. In case there is no entry in the CS matching the Interest, a PIT entry is first created. This entry allows the provider device to serve multiple, concurrently arriving, identical requests with a single message i.e., applying multicast, as in original NDN. In this case, the Requesting Face list of the entry includes the BSSID of the current BSS. Subsequently, the FIB table is looked up and the Internal_Face is used to forward the Interest message to the corresponding application. For completeness, the Internal_Face is also added to the PIT entry as an output face.

4. The response from the application is cached in the CS and sent back to the broadcast domain indicated by the BSSID value of the local PIT entry, which is subsequently removed.

5.7.2 Manual

Tables NDN Forwarding Daemon (NFD) is a network forwarder that implements NDN protocol. We plan to extend NFD reusing most of its components and doing some modifications in order to provide a KEBAPP framework implementation. The tables module provides main data structures for NFD. In the following we describe the main tables used in NDN and the modifications required by KEBAPP:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

- CS: The CS is the cache of the data packets arriving. In this case, we do not need to modify the CS NFD implementation. However, the name used to index the data, will include the name hierarchical prefix name, but also the different hashtags used to define the data included in the name.
- FIB: The FIB is used to forward Interest packets toward potential source(s) of matching Data. It is almost identical to an IP FIB except it allows for a list of outgoing faces rather than a single one. Here, the FIB structure is modified compared with the NDN FIB. The KEBAPP FIB requires to faces: the BSSID of the WiFi network where the application is available, and an internal face with a pointer to the local application that is willing to be shared with other users.
- PIT: The PIT keeps track of Interests forwarded looking for application sharing. In this case, we need to extend the PIT in the same way we extend the FIB, and we need to include a second face, and input face that keeps track of the origin of the Interest (the local application or the WiFI network) and the output face that keeps track of the Interest destination.
- Other tables: Other tables, such as the Network Region Table, the Dead Nonce List or the Interest Table for loop detection do not need to by modified since KEBAPP does not need those tables for its operation. The Measurements Table can be used to store measurements information regarding certain apps.

Managers Connectivity management plays a vital role in KEBAPP. KEBAPP focus on WiFi-enabled (IEEE 802.11) connectivity, including WiFi Direct, which enables mobile devices to act as APs by forming communication groups. The creation of an application-specific BSS requires the ability of mobile devices to identify the mapping between the BSS and the corresponding application. The recently announced WiFi Neighbour Awareness Networking (NAN) protocol can support this requirement. It is noted that a device can be connected to more than one BSSs at the same time, thus acquiring or providing information across several applications.

- KEBAPP WiFi manager: The KEBAPP WiFi manager plays a vital role in KEBAPP. The KEBAPP WiFi Manager is directly connected to the NDN Face Manager and FIB Manager and is responsible at the same time of the link-layer connections to the multiple access networks.
- Face Manager: The Face Manager is responsible of creating and destroying faces. The face manager uses the information provided by the KEBAPP WiFi manager to create or removing faces to the multiple WiFi networks or certain local applications
- FIB Manager: The FIB Manager is responsible of updating the FIB table. FIB manager uses the information provided by the KEBAPP WiFi manager about the networks available in the vicinity and adds new entries for the KEBAPP applications available in each network.

6 Next steps

In the Section above, we have detailed the progress of the project so far towards the development of the UMOBILE architecture. The consortium has worked on a variety of aspects of the UMOBILE architecture, ranging from the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

interconnection of NDN with IBR-DTN to the development of relevant applications and a service migration platform that is able to move services at the edges of the network. The next steps for UMOBILE project include the evolution and optimisation of the developed mechanisms, their integration as a unified UMOBILE platform, as well as extensive testing and evaluation. In particular, the project consortium plans to:

- Perform extensive testing and evaluation of the IBR-DTN – NDN integration in various scenarios;
- Implement the proposed smart routing protocol, as well as a mechanism that creates the relevant entries in the FIB tables;
- Implement Oi! app over NDN and develop one more news app;
- Migrate more complex services and study their performance;
- Integrate push services with service migration platform in order to automatically push services towards the edges of the network;
- Implement the proposed naming scheme and, of course,
- Integrate the different components into a single UMOBILE platform, as described in the UMOBILE vision section.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

7 Conclusion

This document covers the progress of the project so far towards the integration of NDN and IBR-DTN and the development of the unified UMOBILE architecture. In particular, we first present the overall vision of the architecture and then describe in detail the different components that have been developed, their status and the next steps for their integration. For all components that have been deployed, we also attach the relevant code, as well as their manuals.

This is the initial version of the UMOBILE implementation. At this stage, our version includes at-least initial implementation of all critical UMOBILE modules, and in particular, the integration of NDN and IBR-DTN, the migration of a simple web server, the Oi! app and the SOCIO framework, as well as the PerSense Mobile Light service. The implementation so far demonstrates that:

- ICN/DTN integration is feasible indeed. More specific functions and services for a wider range of UMOBILE scenarios will be integrated shortly;
- Service migration is feasible;
- UMOBILE representative applications that incorporate different interfaces, including WiFi direct, have been developed;
- socially-aware opportunistic communication framework has been implemented and can be exploited;
- A service that assists in performing network contextualization has been developed, and
- Push services can be successfully integrated into NDN.

In general, all proposed modules are suitable for the overall target. Further optimisations will be performed during the integration, optimisation and evaluation period. The final integrated version will be submitted to the European Commission as D3.2 on M30.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

8 References

[1] L. Amaral, R. C. Soa, P. Mendes, and W. Moreira, "Oi! – opportunistic data transmission based on Wi-Fi direct," in IEEE Infocom 2016 Live/Video Demonstration (Infocom'16 Demo), (San Francisco, USA), Apr. 2016.

[2] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: Social-based forwarding in delay-tolerant networks," IEEE Transactions on Mobile Computing, vol. 10, no. 11, pp. 1576–1589, November, 2011.

[3] W. Moreira, P. Mendes, and S. Sargento, "Opportunistic routing based on daily routines," in Proceedings of WoWMoM, pp. 1–6, June 2012.

[4] W. Moreira and P. Mendes, "Social-aware opportunistic routing: The new trend," in Routing in Opportunistic Networks (I. Woungang, S. K. Dhurandher, A. Anpalagan, and A. V. Vasilakos, eds.), pp. 27–68, Springer New York, 2013.

[5] W. Moreira and P. Mendes, "Dynamics of social-aware pervasive networks," in Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on, pp. 463–468, March 2015.

[6] W. Moreira and P. Mendes, "Impact of human behavior on social opportunistic forwarding," Ad Hoc Networks, vol. 25, Part B, pp. 293 – 302, 2015. New Research Challenges in Mobile, Opportunistic and Delay-Tolerant Networks Energy-Aware Data Centers: Architecture, Infrastructure, and Communication.

[7] W. Moreira, M. de Souza, P. Mendes, and S. Sargento, "Study on the effect of network dynamics on opportunistic routing," in Ad-hoc, Mobile, and Wireless Networks (X.-Y. Li, S. Papavassiliou, and S. Ruehrup, eds.), vol. 7363 of Lecture Notes in Computer Science, pp. 98–111, Springer Berlin Heidelberg, 2012.

[8] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," SIGMOBILE Mob. Comput. Commun. Rev., vol. 7, pp. 19–20, July 2003.

[9] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco, "Socially-aware routing for publish- subscribe in delay-tolerant mobile ad hoc networks," IEEE J.Sel. A. Commun., vol. 26, no. 5, pp. 748–760, June, 2008.

[10] A. Mtibaa, M. May, C. Diot, and M. Ammar, "Peoplerank: social opportunistic forwarding," in Proceedings of INFOCOM, pp. 111–115, 2010.

[11] H. A. Nguyen and S. Giordano, "Context information prediction for social-based routing in opportunistic networks," Ad Hoc Netw., vol. 10, no. 8, pp. 1557–1569, November, 2012.

[12] W. Moreira, P. Mendes, and S. Sargento, "Social-aware opportunistic routing protocol based on users interactions and interests," in Ad Hoc Networks (M. H. Sherif, A. Mellouk, J. Li, and P. Bellavista, eds.), vol. 129 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 100–115, Springer International Publishing, 2014.

[13] S. Pérez-Sánchez, J. M. Cabero, and I. Urteaga, "Dtn routing optimised by human routines: the hurry protocol," in Wired/Wireless Internet Communications, pp. 299–312, Springer, 2015.

[14] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in Proceedings of CoNEXT, pp. 1–12, December 2009.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

9 Annex 1: Oi! and SOCIO TR

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

Oi! and SOCIO: Opportunistic Data Transmission based on Wi-Fi Direct

Technical Report COPE-SITI-TR-16-03

May 25, 2016

Editor:
Luis A. Lopes (COPELABS/ULHT)

Authors:
Luis A. Lopes (COPELABS/ULHT)
Waldir Moreira (COPELABS/ULHT)
Paulo Mendes (COPELABS/ULHT)
Rute Sofia (COPELABS/ULHT)

Executive Summary

User devices have evolved in terms of processing, storage and built-in wireless communications technologies. Such evolution has allowed users to explore different ways for establishing communication among themselves. This technical report presents the high level design and operation flowcharts of Oi! (an Android messaging application) and SOCIO (a socially-aware opportunistic communication framework) that combined allow users to exchange messages without relying on the existing Internet access or wireless infrastructures, based on their level of social interaction.

Contents

List of Figures

Acronyms

SOCIO	Socially-aware Opportunistic CommunIcation framewOrk
UI	User Interface
CM	Content Manager
IS	Internal Storage
RT	Routing
SW	Social Weight
SP	Social Proximity
TECD	Time-Evolving Contact Duration
DB	Database

1 Introduction

Despite all the technological evolution we have witnessed in the last years concerning device's capabilities and networking paradigms, applications still rely either on the presence of wireless infrastructure or Internet access to allow data exchange among users.

With Oi! [1] and SOCIO[1], we follow a social-based opportunistic networking paradigm to decouple the application from the dependency on the existence of Internet access: by exploiting the direct wireless communications capabilities (i.e., Bluetooth and Wi-Fi direct) available in personal mobile devices, SOCIO measures the social proximity among users, and allows Oi! to exchange messages independently of the availability of Internet access.

Oi! and SOCIO are expected to run over devices carried by users throughout their daily routines. Since these users socially engage in different settings (e.g., home, work, school) and with different others (e.g., family, friends, neighbors, colleagues, classmates), opportunistic data dissemination based on social awareness seems to be a rather interesting approach: Oi! messages are exchanged only between socially well-connected devices (i.e., users with strong social interaction). The reason for such choice refers to the fact that social similarity tends to vary less than node mobility (forwarding table updates are less frequently) and is dynamic enough to allow wise selection of next best forwarders (i.e., intermediate users) according to the social setting in which the involved parties find themselves [2, 3, 4, 5, 6].

Fig. 1 illustrates how Oi! messages are opportunistically exchanged through SOCIO considering the levels of social interaction among users in a working setting.

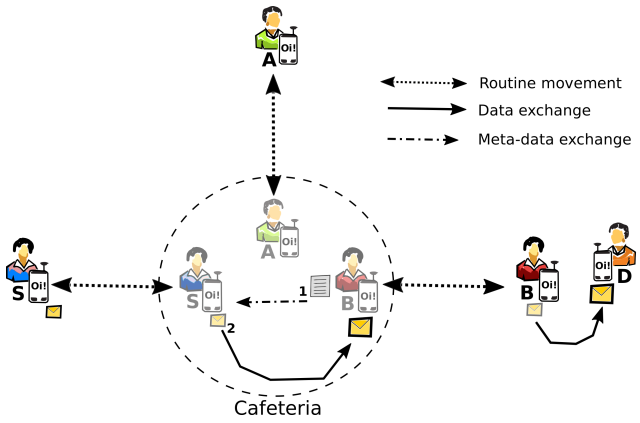


Figure 1: Social-aware opportunistic data transmission.

As aforementioned, users meet one another in different settings and at different levels. In Fig. 1, the source node S has a message to destination node D. During lunch time, node S meets other nodes from which node B happens to work (i.e., spends more time in the same social setting) with node D. By considering the level of interaction of all nodes in the cafeteria towards node D, node B is the best next forwarder to reach it given the social engagement in such social setting.

Based on that, Oi! messages are replicated only to socially relevant nodes (i.e., node B in this example) and refrain from using resources that will not result in the delivery of the message (node A could have received a copy of the message, but such effort would be useless as it does not socially engage with D at all in such setting).

In order to offer a detailed view of Oi! and SOCIO, this technical is structured as follows. Sec. 2 presents the high level design highlighting the developed modules, and explaining the different primitives that allow such modules to interact towards the global functioning of Oi! over the SOCIO framework. Then, Sec. 3 gives a more detailed view on the flowchart operation of Oi! and SOCIO. Finally, Sec. 4 presents the future work to be developed in SOCIO, including the identification some of the limitations inherent to Android that affect the functioning of the Oi! application over SOCIO. In the Annex section, the code documentation for Oi! and SOCIO is provided.

2 Oi! and SOCIO High Level Design

The Oi! application and SOCIO framework follow a modular design, aiming to allow easier extensions, such as the inclusion of other routing schemes, a novel algorithm to infer social proximity, or a new method to manage

[1] https://play.google.com/store/apps/details?id=com.copelabs.oiframework&hl=en

data. As illustrated in Fig. 2, the Oi! application comprises interface module, while the SOCIO framework comprises the remaining modules as explained next.

Oi!'s **User Interface** (UI) module allows the user to compose a message, choose its recipient from a list of known contacts, and to see received messages. The UI interacts with SOCIO's **Content Manager** (CM) module, which is responsible to manage all the messages to be sent, as well as received messages. The primitives *Send(destination, message)*, allows CM to build a *message* to be sent to a given *destination* node; *Receive(message),* allows CM to pass the *received message* to the UI; and *GetContactsList(),* fetches a list of known contacts to display on the UI. It is worth mentioning that a new application (e.g., dissemination app) will make use of these primitives provided by SOCIO's CM.

Based on the data provided by UI, CM starts by composing a new message, which is stored together with the <*sender*, *destination*, *timestamp*> tuple. The sender and destination fields are the deviceID of the message's source and recipient devices, respectively. The timestamp is used to detect possible message duplication. This file is then stored in the Internal Storage (IS). All messages to be sent are stored in toOthers.xml file. Upon getting a new message from the Wi-Fi Direct through the primitive *Receive(received message list)*, CM considers the tuple <*sender*, *destination*, *timestamp*> to filter messages. The filtering process is needed to check: i) whether the same message has been previously received; ii) if the message destination is the device itself, in which case checks if UI is available. If so, CM passes the received message to be shown in UI. Otherwise, CM saves the message to the localCache.xml file so it can be delivered when UI is available.

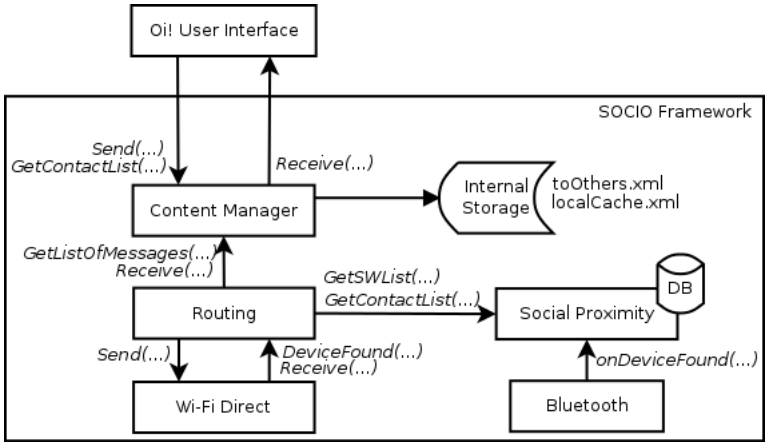


Figure 2: Oi! and SOCIO high level design.

Routing (RT) module implements dLife, a social-aware opportunistic algorithm based on daily routines [3]. Forwarding decisions are based on the level of social interaction among users: message replication only takes place when the social weight of an encountered node towards a specific destination is higher than the social weight of the current carrier of the message towards such destination. When a device is found, through the primitive *DeviceFound(deviceID, SW list)* triggered by the Wi-Fi Direct module, RT gets the identification and list of social weights of the device and by using the primitives *GetListOfMessages(list of destinations)* and *GetSWList(list of destinations),* RT gets a list of carried messages and the social weights towards a specific *list of destinations*, respectively, to help with forwarding decisions. When the forwarding decision is made, RT uses *Send(destination, message list)* to provide the Wi-Fi direct module with a list of messages to be replicated to the *destination* (i.e., encountered Oi! device).

Wi-Fi direct module comprises the notification and data exchange phases. In the notification phase, this module detects the presence of other Oi! devices and notifies RT. As for the data exchange phase, this module i) considers the list of messages from RT to replicate to an encountered Oi! device; and ii) provides CM with a set of messages received from another Oi! device.

Social Proximity (SP) module computes the social weight towards encountered devices through Bluetooth sightings and stores this information on its database. This social weight is given by the Time-Evolving Contact Duration (TECD) [7], which via the primitive *onDeviceFound(deviceID, encounter time)* compares the *encounter time* against the current time after a Bluetooth scan to determine the contact duration towards a specific device (i.e., *deviceID*). Social weight is computed in a hourly base, which can include different contacts of various durations. SP provides the UI with a list of known contacts through CM, and RT with a list containing the potential destination devices and the social weight towards them.

Bluetooth (BT) module is used to detect the presence of neighboring devices. BT provides SP with the identification of the neighbor device and the time of encounter.

3 Oi! and SOCIO Operation Flowcharts

This section presents the detailed operation flowcharts for the Oi! application and SOCIO framework.

3.1 Oi! operation

Oi! works on top of the SOCIO framework to allow users to opportunistically exchange messages based on their level of social engagement. So, the first check (1) is whether the SOCIO framework is available (cf., Fig. 3). If not, the application asks the user to proceed with SOCIO's installation (2). Otherwise, Oi! connects to the framework (3).

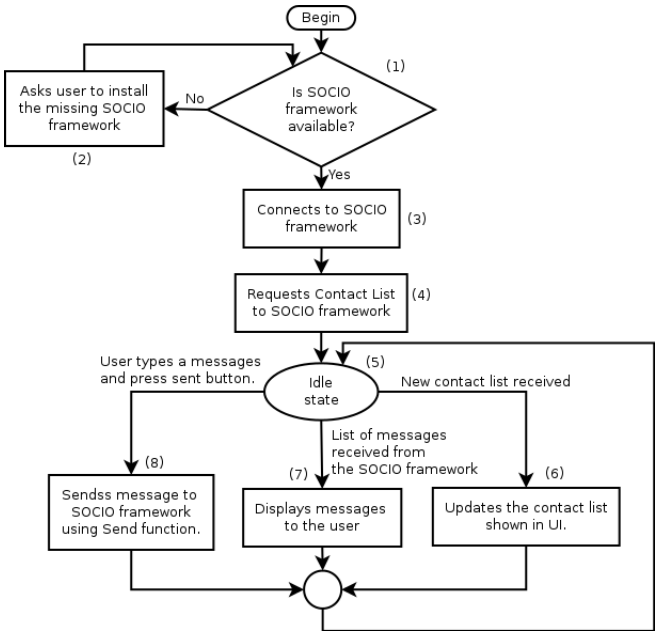


Figure 3: Oi! application operation flowchart.

Once connected to the framework, Oi! requests the contact list (4) and shifts to idles state (5), in which one of the following actions may take place (order is irrelevant and serves for the purpose of explanation only):

- Contact list is received: Oi! then updates the list of contacts on the user interface (6).
- Messages are received: Oi! displays the incoming messages to user (7).
- User composes a message: Oi! forwards the newly composed message to SOCIO framework (8).

3.2 SOCIO operation

SOCIO provides an API, which shall allow for the design of other applications aiming at exploiting social-aware opportunistic communications over multiple wireless technologies regardless of the existence of infrastructure or Internet connectivity. For the sake of simplicity, we explain the operation flowcharts for the SOCIO modules considering the Oi! application.

3.2.1 Content Manager (CM)

CM is the module that provides the primitives that allow Oi! to exploit SOCIO's capabilities. As shown in Fig. 4, this module starts by initializing the TTL verification mechanism (1), which allows the removal of stale data, and the Routing module (2).

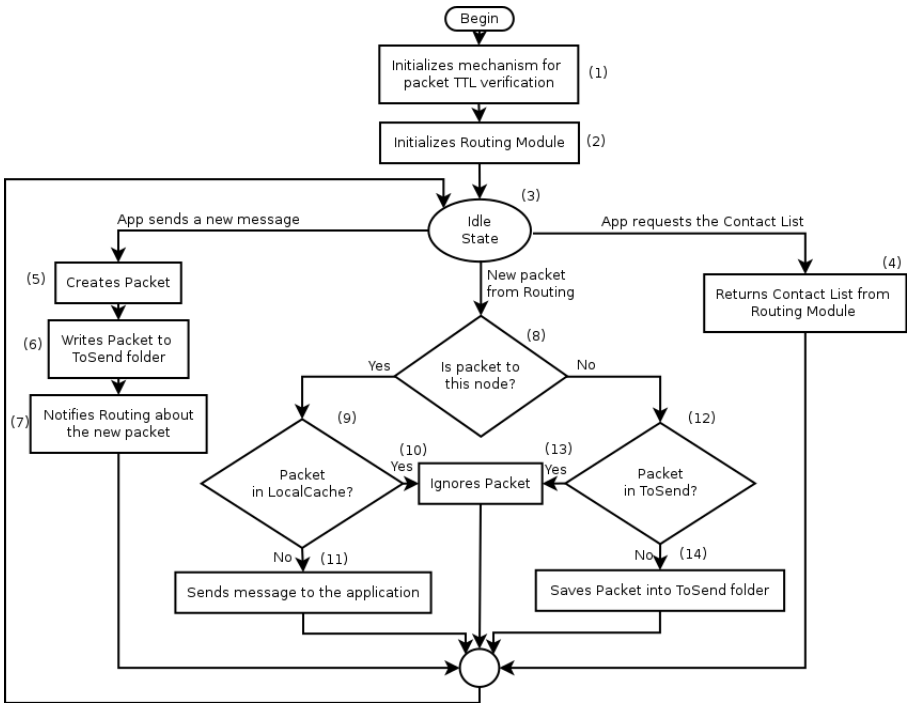


Figure 4: Content Manager operation flowchart.

CM then reaches the idle state (3), in which three possible actions may take place:

- Application request contact list: CM provides the contact list obtained from the routing module (4).
- Application produces an outgoing message: CM creates the packet with message in the payload (5), writes this packet to the outgoing folder - toSend (6), and notifies the routing module on the existence of a new packet to be disseminated (7).
- New packer arrived: CM checks whether the current node is the recipient for this new packet (8). If so, it checks whether the packet has been received before (9). If this packet has been received (already in LocalCache), CM ignores it (10). Otherwise, CM extracts the message from the packet, and sends it to the application (11). In the case the packet is too a different destination, CM checks whether the packet has been received before (already in ToSend, 12). If so, CM ignores the packet (13). Otherwise, it saves the packet in the ToSend folder for further dissemination (14).

3.2.2 Routing (RT)

RT is responsible to deciding whether a packet is forwarded to an encountered device. This decision is made based on the levels of social interactions between the communicating parties and the packet's destination. The exchange of meta-data and packets take place over Wi-Fi.

As illustrated in Fig. 5, RT starts by initializing the Social Proximity (1) and Wi-Fi Direct (2) modules, shifting to an idle state (3).

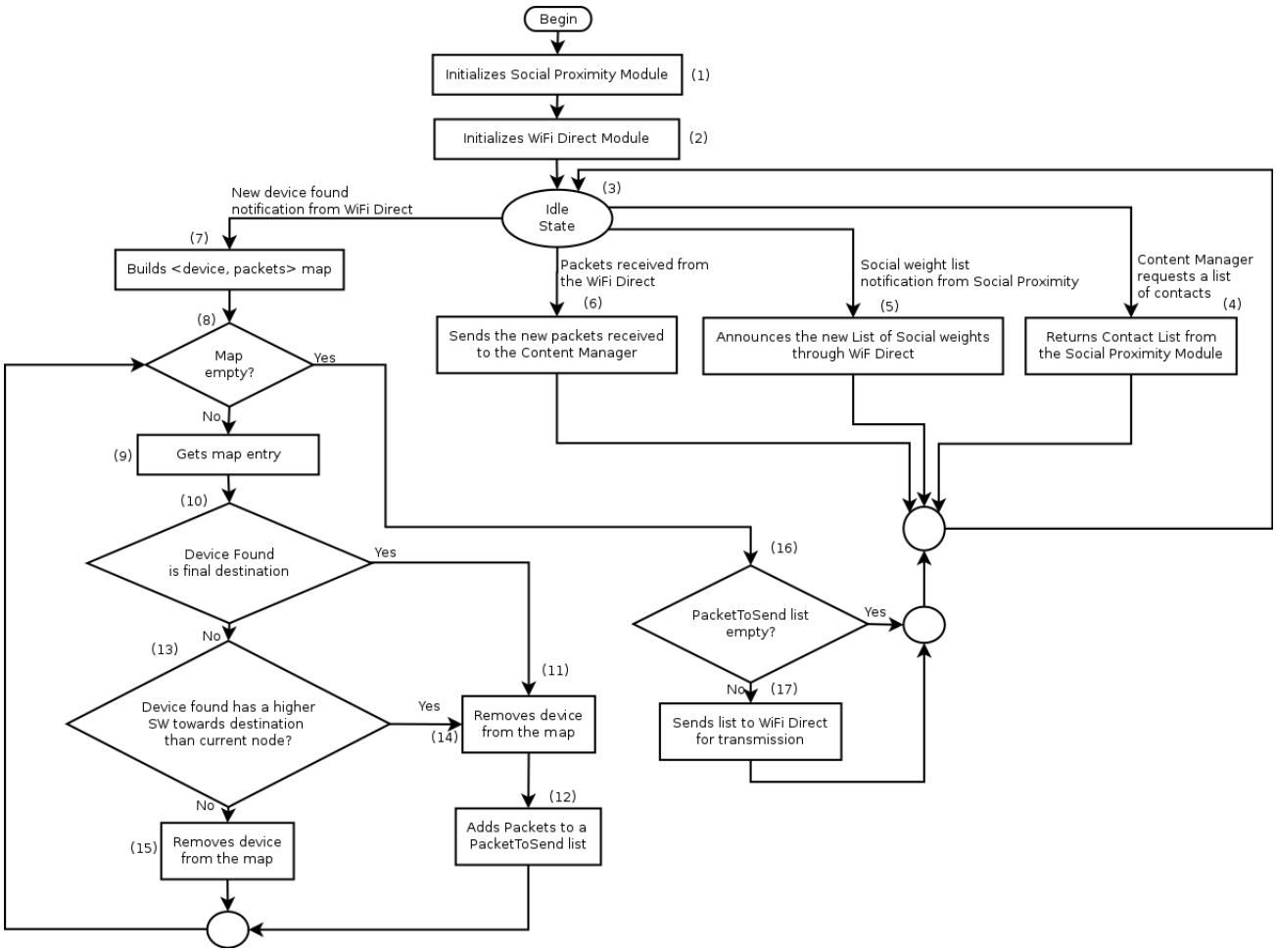


Figure 5: Routing operation flowchart.

While in idle state, RT expects one of the four actions to occur:

- CM requests contact list: RT provides the contact list obtained from the social proximity module (4).
- Social weight list notification received from Social Proximity module: RT announces the received SW list to neighboring peers as this information is used to perform routing decisions (5).
- Packets received from Wi-Fi Direct module: RT sends these packets to CM, which decides if they are new or have already been received (6).
- New device found notification received from Wi-Fi Direct module: RT builds a map (with devices and packets towards them), which includes the found peer, and users that the found peer has listed on its social weight list and that the current node also has on its ToSend folder (7). RT then checks whether the map is empty (8). If not, RT gets the first map entry (9) and check whether the <device> matches the device found (i.e., final destination, 10). If so, RT removes <device> from map (11), and adds the respective packets to the PacketToSend list (12). In the case that <device> does not match the device found, RT checks whether the device found has higher social weight than the current node towards <device> (i.e., destination, 13). If so, RT removes <device> from map (14), and adds the respective packets to the PacketToSend list (12). Otherwise, RT only removes <device> from map (15). This process continues until the map is empty. Then, RT checks whether the PacketToSend list is empty (16). If not, RT sends this list to Wi-Fi Direct module for transmission (17).

3.2.3 Social Proximity

SP is responsible for computing the social weights towards other devices. For that, it relies on Bluetooth scans to detect neighboring devices and keep track of the contact duration between users.

As illustrated in Fig. 6, SP starts by initializing the Bluetooth module (1) and sending it a command to start periodic scans to detected neighboring peers (2). After that, SP, starts the periodic alarm for social weight computation (3), and shifts to idle state (4).

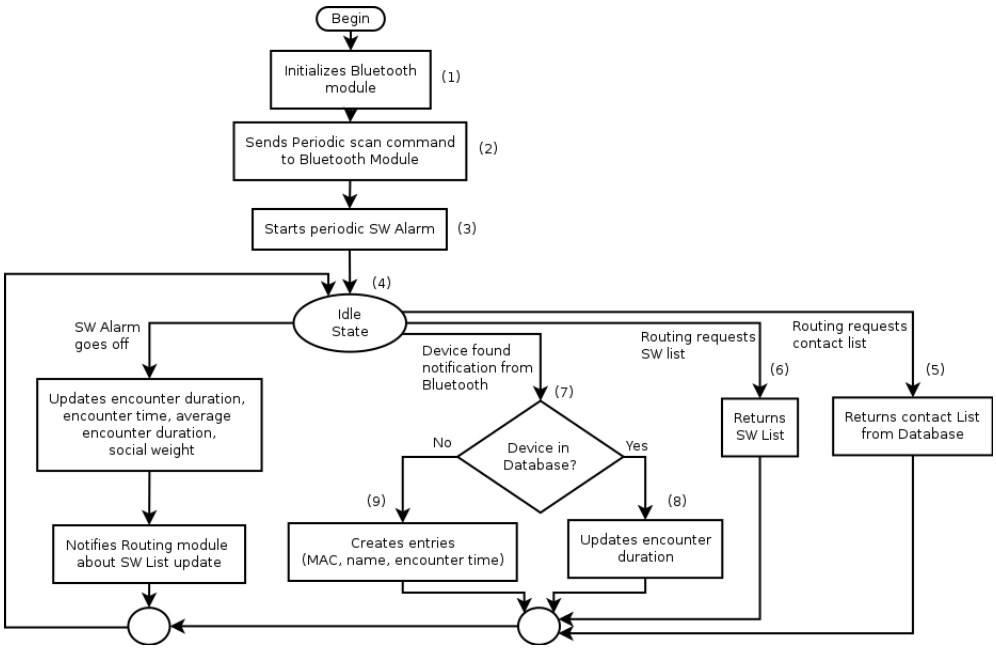


Figure 6: Social Proximity operation flowchart.

While in idle state, SP may receive one of the four actions:

- RT requests contact list: SP returns a list of contacts based on the information of devices added to its Database (5).
- RT requests SW list: this request may come with a set of specific devices (for forwarding decisions), or it may require a complete list (for announcing the current node's social information). Either way, SP responds by returning a list updated social weights (6).
- Device found notification received from Bluetooth module: SP checks whether the found device is already in the Database (7). If so, SP updates encounter duration information for this device (8). Otherwise, the device has not been seen before, and SP then creates entries (9) in the Database with its information (MAC address, name, encounter time).
- Periodic social weight alarm goes off: SP updates encounter and social weight information to all devices it has on the Database (10), and notifies RT about this update (11).

3.2.4 Wi-Fi Direct

Wi-Fi Direct is responsible for transmitting and receiving packets to/from others devices by using the Wi-Fi P2P technology.

As illustrated in Fig. 7, Wi-Fi Direct starts by initializing a Wi-Fi P2P channel that will be used by this module to announce local services, receive neighbor services and also to establish Wi-Fi P2P connections (1). After this initialization process finishes, Wi-Fi Direct shifts to an idle state (2).

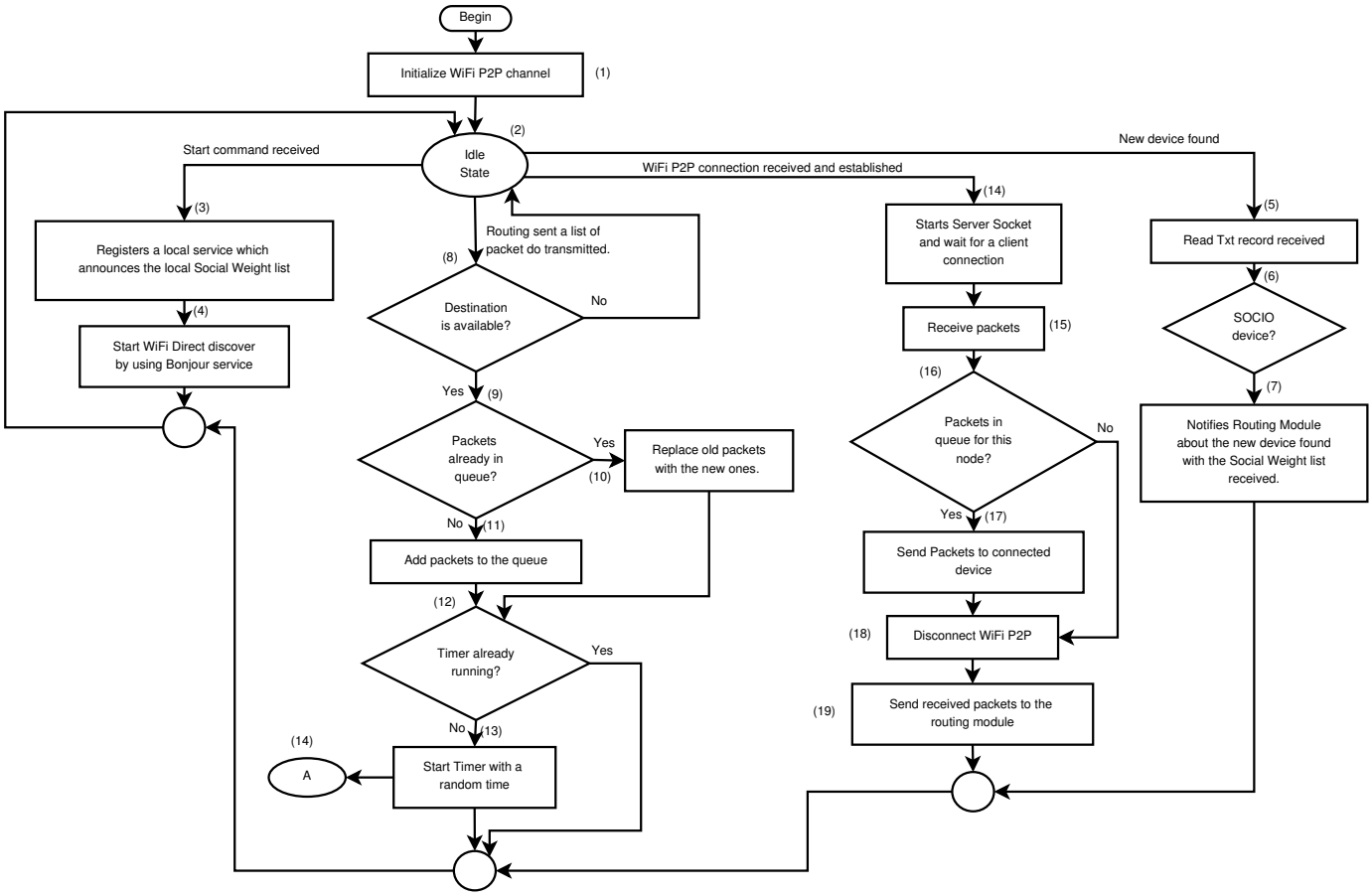


Figure 7: Wi-Fi Direct operation flowchart.

While in idle state, Wi-Fi Direct expects one of the four actions to occur:

- Start command received from the RT: RT provides the contact list obtained from the social proximity module to be announced as local service (3). This start command also initializes the discover process from Bonjour Service (4). This process will detect neighboring devices that have services being announced.
- New device found: After Wi-Fi Direct detects a new device nearby, it reads the TXT record received (5) and checks if there is a SOCIO service available (6). If so, Wi-Fi direct notifies RT about this new device by sending the peer info and the SW list available inside the received TXT record. (7).
- List of packets received from RT: Wi-Fi Direct checks if remote destination is still available (8). If so, Wi-Fi Direct verifies if it already has packets inside the queue to send to the remote device (9) and it replaces the old packets with these new ones (10). If so, it just adds these packets to the queue (11). Otherwise, Wi-Fi Direct ignores these packets since it cannot reach the remote device. By having new content to send, Wi-Fi Direct checks if the timer is currently running (12), starting it if that is not the case (13). The purpose of this timer, further detailed next, is to avoid Wi-Fi collisions (e.g., two devices trying to connect to each other at the same time) by adding a small random delay to each connection. If timer is already running, it means that there is a pending connection and these packets will be transmitted when this pending transmission finishes.
- A Wi-Fi P2P connection received: Wi-Fi Direct module creates a server socket and waits for a client (14). When the client connects to this server socket, the Wi-Fi Direct module receives the packets from the remote device (15). After receiving all packets, Wi-Fi Direct searches the queue for packets to this remote device (16). If there are packets to be sent, then Wi-Fi uses the server socket to transmit these packets to the remote device (17). Otherwise, Wi-Fi Direct closes the server socket and disconnects the Wi-Fi P2P connection (18), and sends the received packets to the RT module (19).

The timer mentioned when Wi-Fi Direct has packets to be sent, is illustrated in Fig. 8.

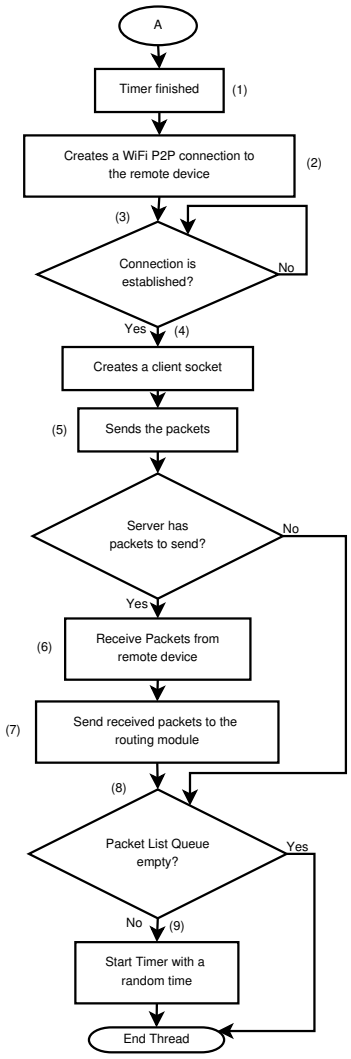


Figure 8: Wi-Fi Direct operation flowchart (thread).

When the timer finishes (1), a Wi-Fi P2P connection is created to the remote device (2). When the Wi-Fi P2P connection is established, Wi-Fi Direct creates a client socket (4) and starts transmitting the packets (5). After transmitting all packets, Wi-Fi Direct checks if the remote device has packets to send. If so, it receives the packets (6) and send them to RT (7). Independently whether the remote device has packet to send, after this process the Wi-Fi Direct checks if the queue is empty (8). If there are still packets to be sent to other remote devices, the Wi-Fi Direct starts again the timer with a small random delay (9). Otherwise, the thread is closed.

3.2.5 Bluetooth

BT is responsible for detecting the presence of neighboring devices for the purposes of social weight computation carried out by the Social Proximity module. As illustrated in Fig. 9, BT starts by initializing the Bluetooth communication interface (1) and registering the framework UUID (2). This UUID is used to identify among the neighboring devices those which are actually running the framework.

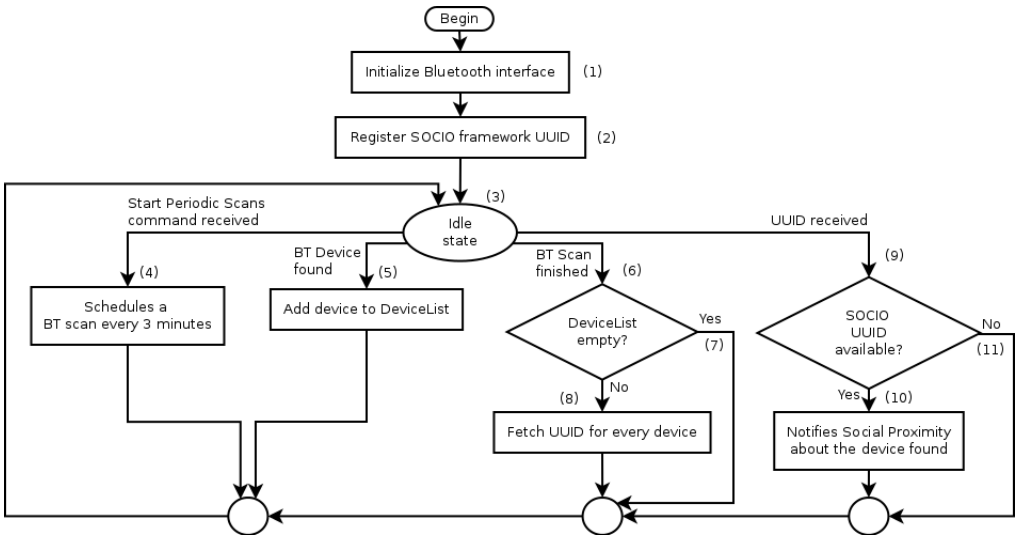


Figure 9: Bluetooth operation flowchart.

Then, BT shifts to the idle state (3), where it expects for one of the four actions to take place:

- Periodic scan command received from social proximity module: BT starts scanning for devices, and sets a scan to be performed at every 3 minutes (4).
- Bluetooth device found: BT adds the found device to the deviceList (5).
- Bluetooth scan finished: BT checks whether the deviceList is empty (6). If so, it goes to idles state (7). Otherwise, BT fetches the service UUIDs for every device in deviceList (8).
- UUIDs received: BT checks whether the SOCIO UUID is among the received ones (9). If so, that means that the device is running SOCIO, and BT notifies the Social Proximity about such device (10). Otherwise, BT shifts back to the idle state (11).

4 Future Work

This section presents the next steps planned for the Oi! application and the SOCIO framework.

4.1 NDN integration

As per current implementation, Oi! and SOCIO follow a host-based communication model. That is, both application and communication framework are designed to allow one-to-one communication. Since the UMOBILE targets content-centric approach, Oi! and SOCIO must be adapted to cope with the content-centric networking paradigm.

The nature of content centricity focus on a pull-based communication model as opposed to the push-based model observed in the host-based paradigm (i.e., sending a content to a specific destination). Thus, we plan to integrate both Oi! and SOCIO into the NDN architecture, allowing our solution to perform one-to-one and one-to-many data dissemination.

This is feasible by means of incorporating extra signaling [8].

4.2 SOCIO Extension

SOCIO will be extended to support a new application, called Now@, aiming to support pervasive data exchange based on users' social interaction and data interested. For that, the new version of SOCIO will integrate, besides dLife [3], a new algorithm called SCORP [9]. SCORP is an opportunistic forwarding algorithm that considers the users' social interaction and their interests to improve data delivery in urban, dense scenarios. While dLife follows a push-based forwarding model, SCORP follows a pull-based forwarding strategy based on the data interests that users inject into the network.

4.3 ACK implementation

So far Oi! and SOCIO are able to produce and disseminate content towards specific destinations nodes. However, the delivery of such content in not confirmed (the only way on knowing if the message arrived is in the case of receiving a response back from the intended destination.

Thus, one improvement that shall be considered is the implementation of acknowledgements for the message that are successfully delivered. In this case the destination node would emit back an ack, which would also be opportunistically exchange among different peers, until the source is notified.

There are implications with that, such as the number of replications. As it happens for the messages, the same would apply to the acknowledgements: a number of replications may occur prior to the due notification of the source, incurring on overhead and unnecessary resource utilization. Still, such mechanism may be interesting (in the case a user is in an emergency situation), and worth considering. Thus, we intend to assess how the usage of ack may be benefitial in improving our solution.

4.4 Limitations to overcome

During the development of Oi! and having in mind opportunistic communications via Wi-Fi, we have encountered several limitations. This subsection explains them, and our workarounds.

4.4.1 Meta-data size constraint during the pre-registration phase

Oi! meta-data comprises a list of MAC addresses of neighboring devices and social weight towards theses devices. This list may grow according to the different devices a user may encounter throughout his/her daily routine. The issue here concerns the amount of information that can be sent during the pre-registration phase (85 bytes) per local service. [10].

Currently, we are analyzing different ways to allow the exchange of more meta-data, such as the use of multiple local services. This requires a mechanism at receiver side to deal with the reception of these multi local services. The current implementation only considers one local service per device.

4.4.2 Wi-Fi direct Authorization Process

In order to establish a direct Wi-Fi connection, it is always required the user authorization for security purposes. This means that every time the devices need to connect to exchange Oi! messages, the communicating users must accept such connection by means of a dialogue box (cf., Fig. 10). Within the context of the Oi! application, this is a drawback as this opportunistic exchange of messages should take place in a seamless manner.

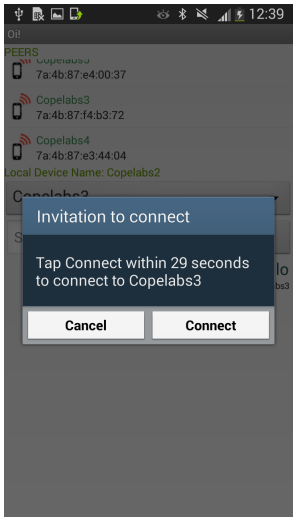


Figure 10: Request to connect.

To resolve this issue we may employ a class which captures the dialogue box and accepts the connection automatically when the connection request is found provided that the application has to be on foreground. This class is a product of Qualcomm[2].

[2]https://github.com/mdabbagh88/alljoyn_java/blob/master/helper/org/alljoyn/bus/p2p/WifiDirectAutoAccept.java

Acknowledgment

The research leading to these results has received funding from the European Unions (EU) Horizon 2020 research and innovation programme under grant agreement No 645124 (Action full title: Universal, mobile-centric and opportunistic communications architecture, Action Acronym: UMOBILE). This technical report reflects only the authors views and the Community is not liable for any use that may be made of the information contained therein.

References

[1] L. Amaral, R. C. Sofia, P. Mendes, and W. Moreira, "Oi! - opportunistic data transmission based on Wi-Fi direct," in *IEEE Infocom 2016 Live/Video Demonstration (Infocom'16 Demo)*, (San Francisco, USA), Apr. 2016.

[2] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: Social-based forwarding in delay-tolerant networks," *Mobile Computing, IEEE Transactions on*, vol. 10, pp. 1576–1589, Nov 2011.

[3] W. Moreira, P. Mendes, and S. Sargento, "Opportunistic routing based on daily routines," in *Proc. of WoWMoM*, pp. 1–6, June 2012.

[4] W. Moreira and P. Mendes, "Social-aware opportunistic routing: The new trend," in *Routing in Opportunistic Networks* (I. Woungang, S. K. Dhurandher, A. Anpalagan, and A. V. Vasilakos, eds.), pp. 27–68, Springer New York, 2013.

[5] W. Moreira and P. Mendes, "Dynamics of social-aware pervasive networks," in *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*, pp. 463–468, March 2015.

[6] W. Moreira and P. Mendes, "Impact of human behavior on social opportunistic forwarding," *Ad Hoc Networks*, vol. 25, Part B, pp. 293 – 302, 2015. New Research Challenges in Mobile, Opportunistic and Delay-Tolerant NetworksEnergy-Aware Data Centers: Architecture, Infrastructure, and Communication.

[7] W. Moreira, M. de Souza, P. Mendes, and S. Sargento, "Study on the effect of network dynamics on opportunistic routing," in *Ad-hoc, Mobile, and Wireless Networks* (X.-Y. Li, S. Papavassiliou, and S. Ruehrup, eds.), vol. 7363 of *Lecture Notes in Computer Science*, pp. 98–111, Springer Berlin Heidelberg, 2012.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of CoNEXT*, pp. 1–12, December 2009.

[9] W. Moreira, P. Mendes, and S. Sargento, "Social-aware opportunistic routing protocol based on users interactions and interests," in *Ad Hoc Networks* (M. H. Sherif, A. Mellouk, J. Li, and P. Bellavista, eds.), vol. 129 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 100–115, Springer International Publishing, 2014.

[10] A. Bhojan and G. W. Tan, "Mumble: Framework for seamless message transfer on smartphones," in *Proceedings of the 1st International Workshop on Experiences with the Design and Implementation of Smart Objects*, SmartObjects '15, (New York, NY, USA), pp. 43–48, ACM, 2015.

Annex

A.1 - Oi! Code Documentation

Oi!

v1.0

Generated by Doxygen 1.8.10

Fri May 20 2016 16:27:47

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

Namespace Documentation

4.1 Package com.copelabs.oiui

Classes

- class ContactListAdapter
- class DeviceListFragment
- class DeviceMessageFragment
- interface FragCommunicatorDevices
- interface FragCommunicatorMessages
- class MainActivity

4.1.1 Detailed Description

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the Oi! Application. This class manages the contact list shown at the activity layout.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the Oi! Application. This class manages a fragment inside the activity that shows the available devices.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the Oi! Application. This class manages a fragment inside the activity that shows the received messages.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the Oi! Application. It provides communication between activity and devices fragment.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the Oi! Application. It provides communication between activity and messages fragment.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the Oi! Application. This class manages the User interface and also triggers the connection to the SOCIO framework.

Author

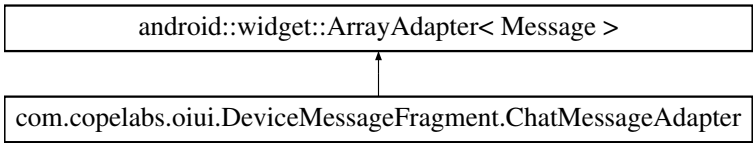
Luis Amaral Lopes (COPELABS/ULHT)

Chapter 5

Class Documentation

5.1 com.copelabs.oiui.DeviceMessageFragment.ChatMessageAdapter Class Reference

Inheritance diagram for com.copelabs.oiui.DeviceMessageFragment.ChatMessageAdapter:



Public Member Functions

- **ChatMessageAdapter** (Context context, int layoutResourceId, List< Message > items)
- View **getView** (int position, View convertView, ViewGroup parent)

5.1.1 Detailed Description

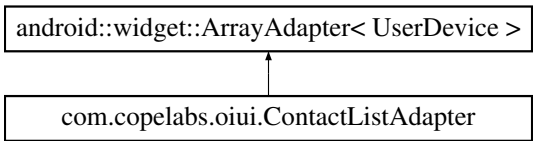
ArrayAdapter to manage chat messages.

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

- src/com/copelabs/oiui/DeviceMessageFragment.java

5.2 com.copelabs.oiui.ContactListAdapter Class Reference

Inheritance diagram for com.copelabs.oiui.ContactListAdapter:



Public Member Functions

- **ContactListAdapter** (Context mContext, List< UserDevice > items)

- View **getDropDownView** (int position, View convertView, ViewGroup parent)
- UserDevice **getItem** (int position)
- View **getView** (int position, View convertView, ViewGroup parent)

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

- src/com/copelabs/oiui/ContactListAdapter.java

5.3 com.copelabs.oiui.DeviceListFragment.DeviceActionListener Interface Reference

Public Member Functions

- void **showDetails** (WifiP2pDevice device)
- void **cancelDisconnect** ()
- void **connect** (WifiP2pConfig config)
- void **disconnect** ()

5.3.1 Detailed Description

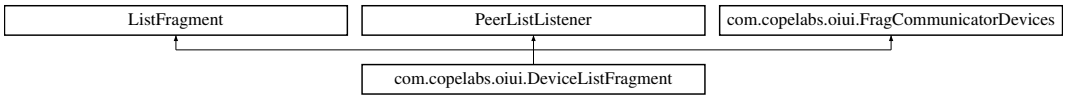
An interface-callback for the activity to listen to fragment interaction events.

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

- src/com/copelabs/oiui/DeviceListFragment.java

5.4 com.copelabs.oiui.DeviceListFragment Class Reference

Inheritance diagram for com.copelabs.oiui.DeviceListFragment:



Classes

- interface DeviceActionListener

Public Member Functions

- void **onAttach** (Activity activity)
- void **onActivityCreated** (Bundle savedInstanceState)
- View **onCreateView** (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
- WifiP2pDevice getDevice ()
- void updateThisDevice (UserDevice device)
- void **removeThisDevice** (UserDevice device)
- void **showMsgFromFramework** (int mError)
- void **onPeersAvailable** (WifiP2pDeviceList peerList)
- void **clearPeers** ()
- void **onInitiateDiscovery** ()
- void **newDeviceInfo** (final UserDevice mDevice)
- void **lostDevice** (final UserDevice mDevice)
- void **showError** (final int mError)

Static Public Attributes

- static final String **TAG** = "DeviceListFragment"

5.4.1 Detailed Description

A ListFragment that displays available peers on discovery and requests the parent activity to handle user interaction events

5.4.2 Member Function Documentation

5.4.2.1 WifiP2pDevice com.copelabs.oiui.DeviceListFragment.getDevice ( )

Returns

this device

5.4.2.2 void com.copelabs.oiui.DeviceListFragment.updateThisDevice (UserDevice *device*)

Update UI for this device.

Parameters

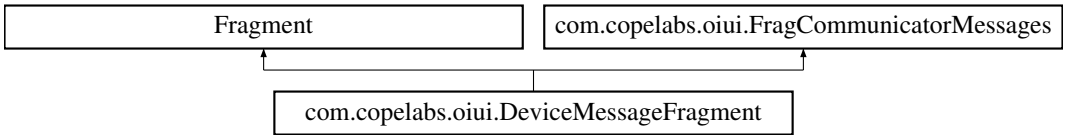
device	WifiP2pDevice object

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

- src/com/copelabs/oiui/DeviceListFragment.java

5.5 com.copelabs.oiui.DeviceMessageFragment Class Reference

Inheritance diagram for com.copelabs.oiui.DeviceMessageFragment:



Classes

- class ChatMessageAdapter

Public Member Functions

- void **onActivityCreated** (Bundle savedInstanceState)
- View **onCreateView** (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
- void **onAttach** (Activity activity)
- void **onActivityResult** (int requestCode, int resultCode, Intent data)
- void showDetails (WifiP2pDevice device)
- void resetViews ()
- void **passDataToFragment** (final UserDevice mSource, final UserDevice mDestination, final String m↩ Message)
- void **pushMessage** (Message mNewMessage)

Static Public Attributes

- static final String **TAG** = "DeviceDetailFragment"
- static WifiP2pInfo **info**
- static boolean **transferStatus** =false

Static Protected Attributes

- static final int **CHOOSE_FILE_RESULT_CODE** = 20

5.5.1 Detailed Description

A fragment that manages a particular peer and allows interaction with device i.e. setting up network connection and transferring data.

5.5.2 Member Function Documentation

5.5.2.1 void com.copelabs.oiui.DeviceMessageFragment.resetViews ( )

Clears the UI fields after a disconnect or direct mode disable operation.

5.5.2.2 void com.copelabs.oiui.DeviceMessageFragment.showDetails (WifiP2pDevice *device*)

Updates the UI with device data

Parameters

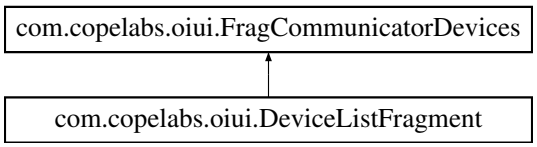
device	the device to be displayed

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

- src/com/copelabs/oiui/DeviceMessageFragment.java

5.6 com.copelabs.oiui.FragCommunicatorDevices Interface Reference

Inheritance diagram for com.copelabs.oiui.FragCommunicatorDevices:



Public Member Functions

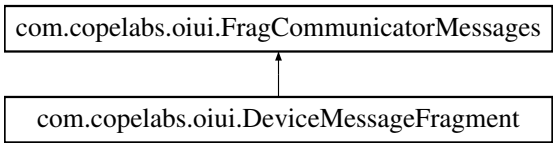
- void **newDeviceInfo** (UserDevice mDevice)
- void **lostDevice** (UserDevice mDevice)
- void **showError** (int mError)

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

- src/com/copelabs/oiui/FragCommunicatorDevices.java

5.7 com.copelabs.oiui.FragCommunicatorMessages Interface Reference

Inheritance diagram for com.copelabs.oiui.FragCommunicatorMessages:



Public Member Functions

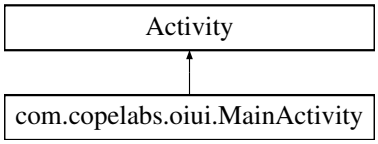
- void **passDataToFragment** (UserDevice mSource, UserDevice mDestination, String mMessage)

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

- src/com/copelabs/oiui/FragCommunicatorMessages.java

5.8 com.copelabs.oiui.MainActivity Class Reference

Inheritance diagram for com.copelabs.oiui.MainActivity:



Classes

- class **RemoteOiFrameworkConnection**

Public Member Functions

- void **onCreate** (Bundle savedInstanceState)
- void onResume ()
- void **onPause** ()
- boolean **onKeyDown** (int keyCode, KeyEvent event)

Public Attributes

- FragCommunicatorMessages fragmentCommunicatorMessages
- FragCommunicatorDevices **fragmentCommunicatorDevices**

Static Public Attributes

- static final String **TAG** = "OiMainActivity"

5.8.1 Member Function Documentation

5.8.1.1 void com.copelabs.oiui.MainActivity.onResume ( )

Register the BroadcastReceiver with the intent values to be matched

5.8.2 Member Data Documentation

5.8.2.1 FragCommunicatorMessages com.copelabs.oiui.MainActivity.fragmentCommunicatorMessages

To comunicate with MessageFragment

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

- src/com/copelabs/oiui/MainActivity.java

A.2 - SOCIO Code Documentation

SOCIO Framework

v1.0

Generated by Doxygen 1.8.10

Fri May 20 2016 16:26:23

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 com

Packages

- package copelabs

5.2 Package com.copelabs

Packages

- package socio

5.3 Package com.copelabs.socio

Packages

- package bt
- package contentmanager
- package router
- package socialproximity
- package wifi

5.4 Package com.copelabs.socio.bt

Classes

- class BluetoothManager
- interface BTDeviceFinder

5.4.1 Detailed Description

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class provides some methods to provide extended functionality to the android BluetoothAdapter.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. It provides the interface between BTManager and SocialProximity.

Author

Waldir Moreira (COPELABS/ULHT)

5.5 Package com.copelabs.socio.contentmanager

Classes

- class CleanTask
- class ContentManager
- class FileIO
- class Packet
- class XmlPullParserHandler

5.5.1 Detailed Description

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. This class verifies the packets TTL. If the TTL is higher than a specific timeout, the packet is removed.

Author

Luis Amaral Lopes (COPELABS/ULHT),

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. Content Manager (CM) module is responsible to manage all of the messages to be sent, as well as received messages. Currently the CM stores messages in an XML format under specific directories.

Author

Luis Amaral Lopes (COPELABS/ULHT),
Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. It provides Write and Read functions to a specific files.

Author

Luis Amaral Lopes (COPELABS/ULHT),
Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. This class represents a packet content.

Author

Luis Amaral Lopes (COPELABS/ULHT),

5.6 Package com.copelabs.socio.router

Classes

- class Routing
- interface RoutingListener

5.6.1 Detailed Description

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. The Routing (RT) module is provided separately to allow relying on any opportunistic forwarding mechanism. The current available forwarding mechanism is based on the dLife algorithm.

Author

Luis Amaral Lopes (COPELABS/ULHT),
Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. It provides the interface between Routing and ContentManager classes.

Author

Luis Amaral Lopes (COPELABS/ULHT),
Waldir Moreira (COPELABS/ULHT)

5.7 Package com.copelabs.socio.socialproximity

Classes

- class DataBase
- interface DataBaseChangeListener
- class OnSocialWeightUpdate
- class SocialProximity
- interface SocialProximityListener
- class SocialWeight
- class SQLiteHelper
- class UserDevAverageEncounterDuration
- class UserDevEncounterDuration
- class UserDeviceInfo
- class UserDevSocialWeight

5.7.1 Detailed Description

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class provides the Database used by Social Proximity class for creation and handling of tables hold information for social weight computation.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. It provides the interface between OnSocialWeightUpdate and SocialProximity.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class is responsible for computing the social weight among users.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class is contains the core functionalities of the application. The BTManager provides all the information from Bluetooth adapter so this class can perform the social context analysis prior to storing the required information in the database.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. It provides the interface between SocialProximity and Routing.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class holds information about a device: name, MAC address, and social weight towards it.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class creates the tables and respective attributes in the database.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class holds the average duration of encounter between a peer and the user device in specific time slots. The information kept are device MAC address and average encounter duration.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class holds the encounter duration between a peer and the user device in specific time slots. The information kept are device MAC address and time spent in the vicinity (i.e., total time within communication range).

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class represents a user device found by means of Bluetooth. The information kept are device name, MAC address, and time of first encounter.

Author

Waldir Moreira (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 06-04-2016 Class is part of the SOCIO application. This class holds the social weight between a peer and the user device in specific time slots. The information kept are device MAC address and social weight.

Author

Waldir Moreira (COPELABS/ULHT)

5.8 Package com.copelabs.socio.wifi

Classes

- class ClientSocketHandler
- class ServerThread
- class WiFiDirect
- class WiFiDirectBroadcastReceiver
- class WiFiDirectDevice
- interface WiFiDirectListener
- class WiFiDirectUtils

5.8.1 Detailed Description

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. Implements the client socket code.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. Server socket code.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. This class is the entry point of the WiFiDirect module. This module is responsible to announce the list of Social Weight, detect new SOCIO enabled devices, send packets to other device and to receive packet from other device. All using WiFi P2P interface.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. A BroadcastReceiver that notifies of important wifi p2p events.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Version

1.0 COPYRIGHTS COPELABS/ULHT, LGPLv3.0, 17-05-2016 Class is part of the SOCIO framework. WiFi↩ Direct listener used to send info/data to modules above.

Author

Luis Amaral Lopes (COPELABS/ULHT)

Chapter 6

Class Documentation

6.1 com.copelabs.socio.bt.BluetoothManager Class Reference

Classes

- class **adapterBroadcastReceiver**
- class **FindBluetoothDevice**

Public Member Functions

- BluetoothManager (Context c)
- void close (Context c)
- void startPeriodicScanning ()
- boolean isBTEnabled ()
- boolean enableBT ()
- boolean startDiscovery ()
- void setOnBTChangeListener (BTDeviceFinder listener)
- void clearOnBTChangeListener ()
- List< String > getLocalInfo ()

Public Attributes

- boolean isScanningActive = false
- boolean isWaitingScanResults = false

Static Public Attributes

- static int DISCOVER_INTERVAL = 180000

Private Member Functions

- void stopPeriodicScanning ()
- void writeToSD (String text)

Private Attributes

- boolean debug = true
- Context mContext
- BluetoothAdapter androidBTAdapter
- BluetoothServerSocket socket
- UUID uuid = UUID.fromString("01010101-0101-0101-0101-010101010101")
- Map< BluetoothDevice, BluetoothClass > btDeviceList = new HashMap<BluetoothDevice, BluetoothClass>()
- FindBluetoothDevice BTDevFinder
- adapterBroadcastReceiver adapterReceiver
- BTDeviceFinder listener
- String fetchingUUIDfromDevice
- Handler mHandler = new Handler()
- Runnable runScan

Static Private Attributes

- static final String TAG = "BTManager"
- static boolean mBTisTurningOnOff = false

6.1.1 Constructor & Destructor Documentation

6.1.1.1 com.copelabs.socio.bt.BluetoothManager.BluetoothManager (Context *c*)

This method is the constructor for BluetoothManager.

Parameters

c	The context.

6.1.2 Member Function Documentation

6.1.2.1 void com.copelabs.socio.bt.BluetoothManager.clearOnBTChangeListener ( )

This method clears the change listener.

6.1.2.2 void com.copelabs.socio.bt.BluetoothManager.close (Context *c*)

This method stops scanning and unregister BroadcastReceivers.

Parameters

c	The context

6.1.2.3 boolean com.copelabs.socio.bt.BluetoothManager.enableBT ( )

This method enables Bluetooth.

6.1.2.4 List<String> com.copelabs.socio.bt.BluetoothManager.getLocalInfo ( )

This method provides the MAC Address and name of local Bluetooth adapter.

Returns

localinfo The local MAC Address

6.1.2.5 boolean com.copelabs.socio.bt.BluetoothManager.isBTEnabled ( )

This method checks whether Bluetooth is enabled.

6.1.2.6 void com.copelabs.socio.bt.BluetoothManager.setOnBTChangeListener (BTDeviceFinder *listener*)

This method sets a change listener.

Parameters

listener	The BTDeviceFinder listener

6.1.2.7 boolean com.copelabs.socio.bt.BluetoothManager.startDiscovery ( )

This method starts Bluetooth discovery process.

6.1.2.8 void com.copelabs.socio.bt.BluetoothManager.startPeriodicScanning ( )

This method starts scanning.

6.1.2.9 void com.copelabs.socio.bt.BluetoothManager.stopPeriodicScanning ( ) [private]

This method stops scanning.

6.1.2.10 void com.copelabs.socio.bt.BluetoothManager.writeToSD (String *text*) [private]

Writes on a file for Bluetooth debugging

6.1.3 Member Data Documentation

6.1.3.1 adapterBroadcastReceiver com.copelabs.socio.bt.BluetoothManager.adapterReceiver [private]

Used to get information from Bluetooth Adapter

6.1.3.2 BluetoothAdapter com.copelabs.socio.bt.BluetoothManager.androidBTAdapter [private]

Used to access functionality of Bluetooth Adapter

6.1.3.3 FindBluetoothDevice com.copelabs.socio.bt.BluetoothManager.BTDevFinder [private]

Used to get information on Bluetooth device

6.1.3.4 Map<BluetoothDevice, BluetoothClass> com.copelabs.socio.bt.BluetoothManager.btDeviceList = new HashMap<BluetoothDevice, BluetoothClass>() [private]

Used to access the Bluetooth device list

6.1.3.5 boolean com.copelabs.socio.bt.BluetoothManager.debug = true [private]

Flag used for debugging purposes

6.1.3.6 int com.copelabs.socio.bt.BluetoothManager.DISCOVER_INTERVAL = 180000 [static]

Sets Bluetooth discover interval

6.1.3.7 String com.copelabs.socio.bt.BluetoothManager.fetchingUUIDfromDevice [private]

Receives UUIDs from Bluetooth device

6.1.3.8 boolean com.copelabs.socio.bt.BluetoothManager.isScanningActive = false

Flag to check if scanning is Active

6.1.3.9 boolean com.copelabs.socio.bt.BluetoothManager.isWaitingScanResults = false

Flag to check if waiting for scan results

6.1.3.10 BTDeviceFinder com.copelabs.socio.bt.BluetoothManager.listener [private]

Used to access the listener of Bluetooth device finder

6.1.3.11 boolean com.copelabs.socio.bt.BluetoothManager.mBTisTurningOnOff = false [static], [private]

Flag to check if Bluetooth is on

6.1.3.12 Context com.copelabs.socio.bt.BluetoothManager.mContext [private]

Used to get global information about an application environment.

6.1.3.13 Handler com.copelabs.socio.bt.BluetoothManager.mHandler = new Handler() [private]

Defines a handler for managing Bluetooth scans

6.1.3.14 Runnable com.copelabs.socio.bt.BluetoothManager.runScan [private]

Initial value:

```
= new Runnable() {
        public void run() {
            if (isScanningActive) {
                if (!isBTEnabled()) {
                    if(enableBT()) {
                        stopPeriodicScanning();
                    } else {
                        Toast.makeText(mContext, "Error enabling BT. Closing Social Pipeline.",
```

```
    Toast.LENGTH_SHORT).show();
                        close(mContext);
                    }
                    return;
                }
                if (!isWaitingScanResults && isBTEnabled()) {
                    btDeviceList.clear();
                    if (startDiscovery()) {
                        if(debug){
                            Log.i(TAG, "Starting BT scaning...");
                            writeToSD("Starting BT scaning...");
                        }
                        isWaitingScanResults = true;
                    }
                    mHandler.postDelayed(runScan,
    DISCOVER_INTERVAL);
                }
                else if (isWaitingScanResults) {
                    mHandler.postDelayed(runScan,
    DISCOVER_INTERVAL);
                    isWaitingScanResults = false;
                }
            }
        }
    }
```

This method allows for starting and stopping periodic Bluetooth scanning.

6.1.3.15 BluetoothServerSocket com.copelabs.socio.bt.BluetoothManager.socket [private]

Sets a listening Bluetooth socket

6.1.3.16 final String com.copelabs.socio.bt.BluetoothManager.TAG = "BTManager" [static], [private]

Tag used for debugging purposes

6.1.3.17 UUID com.copelabs.socio.bt.BluetoothManager.uuid = UUID.fromString("01010101-0101-0101-0101-010101010101") [private]

UUID used to identify Oi! devices

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

- src/com/copelabs/socio/bt/BluetoothManager.java

6.2 com.copelabs.socio.bt.BTDeviceFinder Interface Reference

Inherited by com.copelabs.socio.socialproximity.SocialProximity.ServiceBTListener.

Public Member Functions

- void onDeviceFound (BluetoothDevice device, BluetoothClass btClass)

6.2.1 Member Function Documentation

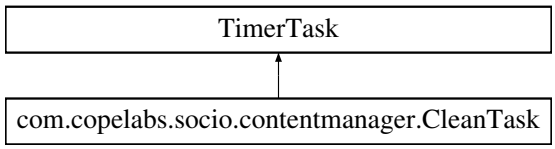
6.2.1.1 void com.copelabs.socio.bt.BTDeviceFinder.onDeviceFound (BluetoothDevice *device*, BluetoothClass *btClass*)

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

- src/com/copelabs/socio/bt/BTDeviceFinder.java

6.3 com.copelabs.socio.contentmanager.CleanTask Class Reference

Inheritance diagram for com.copelabs.socio.contentmanager.CleanTask:



Public Member Functions

- CleanTask ()
- void run ()

Private Member Functions

- List< Packet > checkTTL (List< Packet > mList)

Private Attributes

- String TAG = "TTL_Check"
- int timeout = 86400000

6.3.1 Constructor & Destructor Documentation

6.3.1.1 com.copelabs.socio.contentmanager.CleanTask.CleanTask ( )

CleanTask constructor

6.3.2 Member Function Documentation

6.3.2.1 List<Packet> com.copelabs.socio.contentmanager.CleanTask.checkTTL (List< Packet > *mList*) [private]

Checks if TTL is higher than the timeout.

Parameters

mList	List of packets to perform a TTL check

Returns

A list of packets with only valid TTL.

6.3.2.2 void com.copelabs.socio.contentmanager.CleanTask.run ( )

Reads the packets from the TOSEND folder and checks the TTL of each packet.

6.3.3 Member Data Documentation

6.3.3.1 String com.copelabs.socio.contentmanager.CleanTask.TAG = "TTL_Check" [private]

6.3.3.2 int com.copelabs.socio.contentmanager.CleanTask.timeout = 86400000 [private]

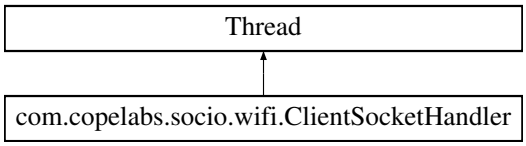
TTL Timeout

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

- src/com/copelabs/socio/contentmanager/CleanTask.java

6.4 com.copelabs.socio.wifi.ClientSocketHandler Class Reference

Inheritance diagram for com.copelabs.socio.wifi.ClientSocketHandler:



Public Member Functions

- ClientSocketHandler (Handler handler, InetAddress groupOwnerAddress, List< Packet > list, String mThis↩ DeviceMACP2p)
- void run ()

Private Attributes

- Handler handler
- InetAddress mAddress
- List< Packet > mDataToSend
- String mThisDeviceMACP2p

Static Private Attributes

- static final String TAG = "ClientSocketHandler"

6.4.1 Constructor & Destructor Documentation

6.4.1.1 com.copelabs.socio.wifi.ClientSocketHandler.ClientSocketHandler (Handler *handler,* InetAddress *groupOwnerAddress,* List< **Packet** > *list,* String *mThisDeviceMACP2p*)

ClientSocketHandler constructor

Parameters

handler	A Handler allows you to send and process Message

groupOwner↩ Address	Server IP address
list	List of packets to send.
mThisDeviceM↩ ACP2p	Local MAC Address

6.4.2 Member Function Documentation

6.4.2.1 void com.copelabs.socio.wifi.ClientSocketHandler.run ( )

6.4.3 Member Data Documentation

6.4.3.1 Handler com.copelabs.socio.wifi.ClientSocketHandler.handler [private]

A Handler allows you to send and process Message

6.4.3.2 InetAddress com.copelabs.socio.wifi.ClientSocketHandler.mAddress [private]

An Internet Protocol (IP) address.

6.4.3.3 List<Packet> com.copelabs.socio.wifi.ClientSocketHandler.mDataToSend [private]

List of packets to send

6.4.3.4 String com.copelabs.socio.wifi.ClientSocketHandler.mThisDeviceMACP2p [private]

Local MAC Address.

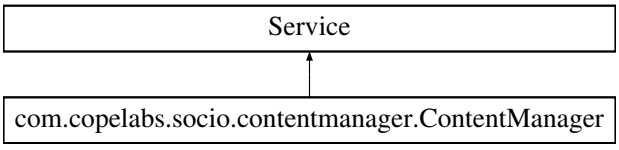
6.4.3.5 final String com.copelabs.socio.wifi.ClientSocketHandler.TAG = "ClientSocketHandler" [static], [private]

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

- src/com/copelabs/socio/wifi/ClientSocketHandler.java

6.5 com.copelabs.socio.contentmanager.ContentManager Class Reference

Inheritance diagram for com.copelabs.socio.contentmanager.ContentManager:



Public Member Functions

- void onCreate ()
- void onDestroy ()
- IBinder onBind (Intent intent)

Private Member Functions

- void initializeModules ()
- void initTTL_Timer ()
- boolean sendPacketToApp (Packet mPacket)
- void receivePacket (List< Packet > mListOfPackets)
- boolean isPacketDuplicated (Packet mNewPacket, String mFile)
- void newDeviceFoundToApp (UserDevice mDevice)
- void deviceLostToApp (UserDevice mDevice)
- void sendError (int mError)
- void contactListUpdatedToApp (List< UserDevice > mList)

Private Attributes

- Routing pRouting
- int interfaceEnabled = 1
- final RemoteCallbackList< IRemoteOiFrameworkCallback > mCallbacks = new RemoteCallbackList <I↩ RemoteOiFrameworkCallback> ()
- final IRemoteOiFramework.Stub mBinder

Static Private Attributes

- static final String TAG = "SOCIO_ContentManager"
- static final String TAG2 = "Oi Demo"

6.5.1 Member Function Documentation

6.5.1.1 void com.copelabs.socio.contentmanager.ContentManager.contactListUpdatedToApp (List< UserDevice > *mList*) [private]

Sends a list of contacts, provided by the Social Proximity.

Parameters

mList	- List of contacts

6.5.1.2 void com.copelabs.socio.contentmanager.ContentManager.deviceLostToApp (UserDevice *mDevice*) [private]

Notifies registered applications about a device that is no longer available.

Parameters

mDevice	

6.5.1.3 void com.copelabs.socio.contentmanager.ContentManager.initializeModules ( ) [private]

Routing module is initialized inside this function. This method also implements the listeners used by the Routing module to communicate with the Content Manager.

6.5.1.4 void com.copelabs.socio.contentmanager.ContentManager.initTTL_Timer ( ) [private]

Initialize the timer that checks the packets TTL. It runs every hour.

6.5.1.5 boolean com.copelabs.socio.contentmanager.ContentManager.isPacketDuplicated (Packet *mNewPacket,* String *mFile*) [private]

Check if the new packet received is already saved inside the Local Cache file

Parameters

mNewPacket	Packet to check if it this duplicated
mFile	File to check.

Returns

true if is already available at local cache or false if not.

6.5.1.6 void com.copelabs.socio.contentmanager.ContentManager.newDeviceFoundToApp (UserDevice *mDevice*) [private]

Notifies the registered applications about a new device found.

Parameters

mDevice	Info about the device found.

6.5.1.7 IBinder com.copelabs.socio.contentmanager.ContentManager.onBind (Intent *intent*)

6.5.1.8 void com.copelabs.socio.contentmanager.ContentManager.onCreate ( )

6.5.1.9 void com.copelabs.socio.contentmanager.ContentManager.onDestroy ( )

6.5.1.10 void com.copelabs.socio.contentmanager.ContentManager.receivePacket (List< Packet > *mListOfPackets*) [private]

Method used when a new packet arrived. It checks if the packet is for this device. If it is, tries to send to the correct app. If not saves the packet to a file.

Parameters

mListOfPackets	List of packets received.

6.5.1.11 void com.copelabs.socio.contentmanager.ContentManager.sendError (int *mError*) [private]

Notifies registered applications about a error occurred inside the Framework.

Parameters

mError	Error code

6.5.1.12 boolean com.copelabs.socio.contentmanager.ContentManager.sendPacketToApp (Packet *mPacket*) [private]

Method the tries to send the packet to the correct app.

Parameters

mPacket	Packet to be sent.

Returns

true if packet was successfully sent, false if app is not available (callback not registered)

6.5.2 Member Data Documentation

6.5.2.1 int com.copelabs.socio.contentmanager.ContentManager.interfaceEnabled = 1 [private]

Flag about the interface state

6.5.2.2 final IRemoteOiFramework.Stub com.copelabs.socio.contentmanager.ContentManager.mBinder [private]

Implements the function that can be used by the registered applications (e.g. Oi!).

6.5.2.3 final RemoteCallbackList<IRemoteOiFrameworkCallback> com.copelabs.socio.contentmanager.↩ ContentManager.mCallbacks = new RemoteCallbackList <IRemoteOiFrameworkCallback> () [private]

Callback list used to send data to the registered applications (e.g. Oi!)

6.5.2.4 Routing com.copelabs.socio.contentmanager.ContentManager.pRouting [private]

Routing module

6.5.2.5 final String com.copelabs.socio.contentmanager.ContentManager.TAG = "SOCIO_ContentManager" [static], [private]

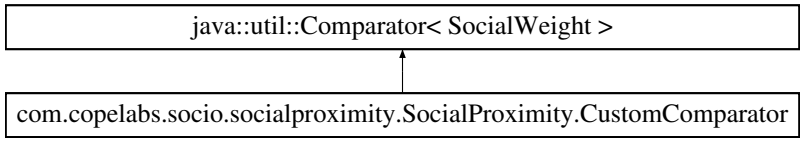
6.5.2.6 final String com.copelabs.socio.contentmanager.ContentManager.TAG2 = "Oi Demo" [static], [private]

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

- src/com/copelabs/socio/contentmanager/ContentManager.java

6.6 com.copelabs.socio.socialproximity.SocialProximity.CustomComparator Class Reference

Inheritance diagram for com.copelabs.socio.socialproximity.SocialProximity.CustomComparator:



Public Member Functions

- int compare (SocialWeight entry1, SocialWeight entry2)

6.6.1 Detailed Description

This class allows to sort social weight entries in ascending order.

6.6.2 Member Function Documentation

6.6.2.1 int com.copelabs.socio.socialproximity.SocialProximity.CustomComparator.compare (SocialWeight *entry1,* SocialWeight *entry2*)

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

- src/com/copelabs/socio/socialproximity/SocialProximity.java

6.7 com.copelabs.socio.socialproximity.DataBase Class Reference

Public Member Functions

- DataBase (Context context)
- void openDB (boolean writable) throws SQLException
- void closeDB ()
- long getNumBTDevice ()
- void registerNewBTDevice (UserDeviceInfo btDev, UserDevEncounterDuration duration, UserDevAverageEncounterDuration averageDuration, UserDevSocialWeight socialWeight)
- void updateBTDeviceAndDuration (UserDeviceInfo btDev, UserDevEncounterDuration duration)
- void updateBTDevAvgEncounterDuration (UserDevAverageEncounterDuration averageDuration)
- void updateBTDevSocialWeight (UserDevSocialWeight socialWeight)
- UserDeviceInfo getBTDevice (String btDevice)
- UserDevEncounterDuration getBTDeviceEncounterDuration (String btDevEncounterDuration)
- UserDevAverageEncounterDuration getBTDeviceAverageEncounterDuration (String btDevAverageEncounterDuration)
- UserDevSocialWeight getBTDeviceSocialWeight (String btDevSocialWeight)
- boolean hasBTDevice (String btDev)
- Map< String, UserDeviceInfo > getAllBTDevice ()
- Map< String, UserDevEncounterDuration > getAllBTDevEncounterDuration ()
- Map< String, UserDevAverageEncounterDuration > getAllBTDevAverageEncounterDuration ()
- Map< String, UserDevSocialWeight > getAllBTDevSocialWeight ()

Private Member Functions

- UserDeviceInfo cursorToBTDevice (Cursor cursor)
- UserDevEncounterDuration cursorToBTDevEncounterDuration (Cursor cursor)
- UserDevAverageEncounterDuration cursorToBTDevAverageEncounterDuration (Cursor cursor)
- UserDevSocialWeight cursorToBTDevSocialWeight (Cursor cursor)

Private Attributes

- SQLiteDatabase db
- boolean isDbOpen
- SQLiteHelper dbHelper
- String[ ] allColumnsBTDevices
- String[ ] allColumnsBTDeviceEncounterDuration
- String[ ] allColumnsBTDeviceAverageEncounterDuration
- String[ ] allColumnsBTDeviceSocialWeight

6.7.1 Detailed Description

This class provides the Database used by Social Proximity class for creation and handling of tables hold information for social weight computation.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 com.copelabs.socio.socialproximity.DataBase.DataBase (Context *context*)

This method is the constructor for DataBase.

Parameters

context	The context.

6.7.3 Member Function Documentation

6.7.3.1 void com.copelabs.socio.socialproximity.DataBase.closeDB ( )

This method closes the predefined database.

6.7.3.2 UserDevAverageEncounterDuration com.copelabs.socio.socialproximity.DataBase.cursorToBTDevAverageEncounterDuration (Cursor *cursor*) [private]

This method converts a cursor pointing to a record in the BTDEVICEAVERAGEENCOUNTERDURATION table to a BTUserDevAverageEncounterDuration object.

Parameters

cursor	The cursor pointing to a record of the BTDEVICEAVERAGEENCOUNTERDURATION table.

Returns

averageDuration The BTUserDevAverageEncounterDuration object.

6.7.3.3 UserDevEncounterDuration com.copelabs.socio.socialproximity.DataBase.cursorToBTDevEncounterDuration (Cursor *cursor*) [private]

This method converts a cursor pointing to a record in the BTDEVICEENCOUNTERDURATION table to a BTUserDevEncounterDuration object.

Parameters

cursor	The cursor pointing to a record of the BTDEVICEENCOUNTERDURATION table.

Returns

duration The BTUserDevEncounterDuration object.

6.7.3.4 UserDeviceInfo com.copelabs.socio.socialproximity.DataBase.cursorToBTDevice (Cursor *cursor*) [private]

This method converts a cursor pointing to a record in the BTDEVICE table to a BTDevice object.

Parameters

cursor	The cursor pointing to a record of the BTDEVICE table.

Returns

btDev The BTDevice object

6.7.3.5 UserDevSocialWeight com.copelabs.socio.socialproximity.DataBase.cursorToBTDevSocialWeight (Cursor *cursor*) [private]

This method converts a cursor pointing to a record in the BTDEVICESOCIALWEIGHT table to a BTUserDevSocialWeight object.

Parameters

cursor	The cursor pointing to a record of the BTDEVICEAVERAGEENCOUNTERDURATION table.

Returns

socialWeight The BTUserDevSocialWeight object.

6.7.3.6 Map<String, UserDevAverageEncounterDuration> com.copelabs.socio.socialproximity.DataBase.getAllBTDevAverageEncounterDuration ( )

This method gets the all the BTUserDevAverageEncounterDuration recorded by the application on the BTDEVICEAVERAGEENCOUNTERDURATION table.

Returns

btDevAverageEncounterDurationMap The map with the BTUserDevAverageEncounterDuration objects, and the BTDEV_MAC_ADDRESS as key.

6.7.3.7 Map<String, UserDevEncounterDuration> com.copelabs.socio.socialproximity.DataBase.getAllBTDevEncounterDuration ( )

This method gets the all the BTUserDevEncounterDuration recorded by the application on the BTDEVICEENCOUNTERDURATION table.

Returns

btDevEncounterDurationMap The map with the BTUserDevEncounterDuration objects, and the BTDEV_MAC_ADDRESS as key.

6.7.3.8 Map<String, UserDeviceInfo> com.copelabs.socio.socialproximity.DataBase.getAllBTDevice ( )

This method gets the all the BTDevice recorded by the application on the BTDEVICE table.

Returns

The map with the BTUserDevice objects, and the BTDEV_MAC_ADDRESS as key.

6.7.3.9 Map<String, UserDevSocialWeight> com.copelabs.socio.socialproximity.DataBase.getAllBTDevSocialWeight ( )

This method gets the all the BTUserDevSocialWeight recorded by the application on the BTDEVICESOCIALWEI↩ GHT table.

Returns

btDevSocialWeightMap The map with the BTUserDevSocialWeight objects, and the BTDEV_MAC_ADDRE↩ SS as key.

6.7.3.10 UserDeviceInfo com.copelabs.socio.socialproximity.DataBase.getBTDevice (String *btDevice*)

This method gets information about a BTDevice already registered by the application.

Parameters

btDevice	The MAC address of the BTDevice which information should be returned.

Returns

The btDev object, null if not found.

6.7.3.11 UserDevAverageEncounterDuration com.copelabs.socio.socialproximity.DataBase.getBTDeviceAverage↩ EncounterDuration (String *btDevAverageEncounterDuration*)

This method gets average encounter duration information of a BTDevice already registered by the application.

Parameters

btDevAverage↩ Encounter↩ Duration	The MAC address of the BTDevice which information should be returned.

Returns

The btDevAvgEncDur object, null if not found.

6.7.3.12 UserDevEncounterDuration com.copelabs.socio.socialproximity.DataBase.getBTDeviceEncounterDuration (String *btDevEncounterDuration*)

This method gets encounter duration information of a BTDevice already registered by the application.

Parameters

btDev↩ Encounter↩ Duration	The MAC address of the BTDevice which information should be returned.

Returns

The btDevEncDur object, null if not found.

6.7.3.13 UserDevSocialWeight com.copelabs.socio.socialproximity.DataBase.getBTDeviceSocialWeight (String *btDevSocialWeight*)

This method gets social weight information of a BTDevice already registered by the application.

Parameters

btDevSocial⇔ Weight	The MAC address of the BTDevice which information should be returned.

Returns

The btDevSocWeight object, null if not found.

6.7.3.14 long com.copelabs.socio.socialproximity.DataBase.getNumBTDevice ( )

This method gets the number of records in the BTDEVICE table. This is, the number of BTDevice registered on the application.

Returns

The number of BTDevice registered by the application.

6.7.3.15 boolean com.copelabs.socio.socialproximity.DataBase.hasBTDevice (String *btDev*)

This method checks if a given BTDevice has already been registered by the application.

Parameters

btDev	The MAC address of the BTDevice.

Returns

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

6.7.3.16 void com.copelabs.socio.socialproximity.DataBase.openDB (boolean *writable*) throws SQLException

This method opens the predefined database.

Parameters

writable	The flag to determine if database is writable

Exceptions

SQLException	

6.7.3.17 void com.copelabs.socio.socialproximity.DataBase.registerNewBTDevice (UserDeviceInfo *btDev,* UserDevEncounterDuration *duration,* UserDevAverageEncounterDuration *averageDuration,* UserDevSocialWeight *socialWeight*)

This method registers a new BTDevice in the application. It creates a new record on the BTDEVICE, BTDEV⇔ ICEENCOUNTERDURATION, BTDEVICEAVERAGEENCOUNTERDURATION, and BTDEVICESOCIALWEIGHT tables, with the information passed as BTDevice.

Parameters

btDev	The Bluetooth device information.
duration	The Bluetooth device information regarding the duration that the BT device is within communication range of others.
averageDuration	The Bluetooth device information regarding the average duration of encounter between the BT device and other devices.
socialWeight	The Bluetooth device information regarding the social weight of the BT device towards others.

6.7.3.18 void com.copelabs.socio.socialproximity.DataBase.updateBTDevAvgEncounterDuration (UserDevAverageEncounterDuration *averageDuration*)

This method updates a BTDevice already registered by the application. This modifies the corresponding record to the BTDevice in the BTDEVICEAVERAGEENCOUNTERDURATION table.

Parameters

averageDuration	The Bluetooth device information regarding its average encounter duration.

6.7.3.19 void com.copelabs.socio.socialproximity.DataBase.updateBTDeviceAndDuration (UserDeviceInfo *btDev,* UserDevEncounterDuration *duration*)

This method updates a BTDevice already registered by the application. This modifies the corresponding record to the BTDevice in the BTDEVICE, BTDEVICEENCOUNTERDURATION, BTDEVICEAVERAGEENCOUNTERDU↩RATION, and BTDEVICESOCIALWEIGHT tables.

Parameters

btDev	The Bluetooth device information.
duration	The Bluetooth device information regarding the duration that the BT device is within communication range of others.

6.7.3.20 void com.copelabs.socio.socialproximity.DataBase.updateBTDevSocialWeight (UserDevSocialWeight *socialWeight*)

This method updates a BTDevice already registered by the application. This modifies the corresponding record to the BTDevice in the BTDEVICESOCIALWEIGHT table.

Parameters

socialWeight	The Bluetooth device information regarding its social weight.

6.7.4 Member Data Documentation

6.7.4.1 String [ ] com.copelabs.socio.socialproximity.DataBase.allColumnsBTDeviceAverageEncounterDuration [private]

Initial value:

```
= {
        SQLiteHelper.COLUMN_BTDEV_MAC_ADDRESS,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT1,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT2,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT3,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT4,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT5,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT6,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT7,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT8,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT9,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT10,
```

```
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT11,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT12,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT13,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT14,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT15,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT16,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT17,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT18,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT19,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT20,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT21,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT22,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT23,
        SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT24
    }
```

List of all columns on the BTDEVICEAVERAGEENCOUNTERDURATION table.

6.7.4.2 String [ ] com.copelabs.socio.socialproximity.DataBase.allColumnsBTDeviceEncounterDuration [private]

Initial value:

```
= {
        SQLiteHelper.COLUMN_BTDEV_MAC_ADDRESS,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT1,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT2,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT3,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT4,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT5,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT6,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT7,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT8,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT9,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT10,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT11,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT12,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT13,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT14,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT15,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT16,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT17,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT18,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT19,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT20,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT21,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT22,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT23,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT24
    }
```

List of all columns on the BTDEVICEENCOUNTERDURATION table.

6.7.4.3 String [ ] com.copelabs.socio.socialproximity.DataBase.allColumnsBTDevices [private]

Initial value:

```
= {
        SQLiteHelper.COLUMN_BTDEV_MAC_ADDRESS,
        SQLiteHelper.COLUMN_BTDEV_NAME,
        SQLiteHelper.COLUMN_BTDEV_ENCOUNTERSTART
    }
```

List of all columns on the BTDEVICE table.

6.7.4.4 String [ ] com.copelabs.socio.socialproximity.DataBase.allColumnsBTDeviceSocialWeight [private]

Initial value:

```
= {
        SQLiteHelper.COLUMN_BTDEV_MAC_ADDRESS,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT1,
```

```
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT2,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT3,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT4,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT5,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT6,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT7,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT8,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT9,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT10,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT11,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT12,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT13,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT14,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT15,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT16,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT17,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT18,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT19,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT20,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT21,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT22,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT23,
        SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT24
    }
```

List of all columns on the BTDEVICESOCIALWEIGHT table.

6.7.4.5 SQLiteDatabase com.copelabs.socio.socialproximity.DataBase.db [private]

Used to handle database

6.7.4.6 SQLiteHelper com.copelabs.socio.socialproximity.DataBase.dbHelper [private]

Used for creating tables and their attributes in the database

6.7.4.7 boolean com.copelabs.socio.socialproximity.DataBase.isDbOpen [private]

Flag used to check whether database is open

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

- src/com/copelabs/socio/socialproximity/DataBase.java

6.8 com.copelabs.socio.socialproximity.DataBaseChangeListener Interface Reference

Public Member Functions

- void onDataBaseChangeUserDevice (ArrayList< UserDeviceInfo > arrayList)
- void onDataBaseChangeEncDur (ArrayList< UserDevEncounterDuration > arrayList)
- void onDataBaseChangeAvgEncDur (ArrayList< UserDevAverageEncounterDuration > arrayList)
- void onDataBaseChangeSocialWeight (ArrayList< UserDevSocialWeight > arrayList)

6.8.1 Member Function Documentation

6.8.1.1 void com.copelabs.socio.socialproximity.DataBaseChangeListener.onDataBaseChangeAvgEncDur (ArrayList< UserDevAverageEncounterDuration > *arrayList*)

6.8.1.2 void com.copelabs.socio.socialproximity.DataBaseChangeListener.onDataBaseChangeEncDur (ArrayList< UserDevEncounterDuration > *arrayList*)

6.8.1.3 void com.copelabs.socio.socialproximity.DataBaseChangeListener.onDataBaseChangeSocialWeight (ArrayList< UserDevSocialWeight > *arrayList*)

6.8.1.4 void com.copelabs.socio.socialproximity.DataBaseChangeListener.onDataBaseChangeUserDevice (ArrayList< UserDeviceInfo > *arrayList*)

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

- src/com/copelabs/socio/socialproximity/DataBaseChangeListener.java

6.9 com.copelabs.socio.contentmanager.FileIO Class Reference

Static Public Member Functions

- static boolean writeFile (String mFileType, Packet mPacket)
- static boolean writeListFile (String mFileType, List< Packet > mList, boolean mAppend)
- static List< Packet > readFile (String mFileType)

Static Public Attributes

- static final String TOSEND = "toSend"
- static final String LOCALCACHE = "localCache"

Static Private Attributes

- static final String TAG = "FileIO"
- static final String mOthersFolder = "/Oi2.0/ToSend/"
- static final String mOthersFile = "toSend.xml"
- static final String mNodeFolder = "/Oi2.0/LocalCache/"
- static final String mNodeFile = "localCache.xml"

6.9.1 Member Function Documentation

6.9.1.1 static List<Packet> com.copelabs.socio.contentmanager.FileIO.readFile (String *mFileType*) [static]

Reads a list of packets at the file chosen by the caller.

Parameters

mFileType	- List of Packets

6.9.1.2 static boolean com.copelabs.socio.contentmanager.FileIO.writeFile (String *mFileType*, Packet *mPacket*) [static]

Writes the packet to an specific xml file in internal storage (private) The caller choose which file it want to write.

Parameters

mFileType	- The file where the packet must be written.

mPacket	- The packet

6.9.1.3 static boolean com.copelabs.socio.contentmanager.FileIO.writeListFile (String *mFileType,* List< Packet > *mList,* boolean *mAppend*) [static]

Writes a list of packets to be sent to a specific xml file in internal storage (private) The caller choose which file it want to write.

Parameters

mFileType	- The file where the list of packets must be written.
mList	- The list of packets.
mAppend	- If append is true and the file already exists, it will be appended to; otherwise it will be truncated.

6.9.2 Member Data Documentation

6.9.2.1 final String com.copelabs.socio.contentmanager.FileIO.LOCALCACHE = "localCache" [static]

6.9.2.2 final String com.copelabs.socio.contentmanager.FileIO.mNodeFile = "localCache.xml" [static], [private]

"LocalCache" file name

6.9.2.3 final String com.copelabs.socio.contentmanager.FileIO.mNodeFolder = "/Oi2.0/LocalCache/" [static], [private]

/** Path to the "LocalCache" folder

6.9.2.4 final String com.copelabs.socio.contentmanager.FileIO.mOthersFile = "toSend.xml" [static], [private]

"ToSend" file name

6.9.2.5 final String com.copelabs.socio.contentmanager.FileIO.mOthersFolder = "/Oi2.0/ToSend/" [static], [private]

Path to the "ToSend" folder

6.9.2.6 final String com.copelabs.socio.contentmanager.FileIO.TAG = "FileIO" [static], [private]

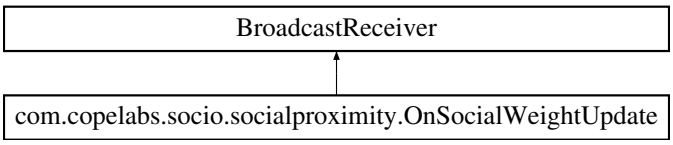
6.9.2.7 final String com.copelabs.socio.contentmanager.FileIO.TOSEND = "toSend" [static]

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

- src/com/copelabs/socio/contentmanager/FileIO.java

6.10 com.copelabs.socio.socialproximity.OnSocialWeightUpdate Class Reference

Inheritance diagram for com.copelabs.socio.socialproximity.OnSocialWeightUpdate:



Public Member Functions

- OnSocialWeightUpdate (DataBase database2, SocialProximity callback)
- void onReceive (Context context, Intent intent)
- void computeSocialWeight (int savDay, int currDay, int currTimeSlot, long currTime, boolean appR, boolean updOverDiffDays)
- int getTimeSlot ()
- void showDevicesOnDB ()

Static Public Attributes

- static int day = 1
- static final String NEW_HOUR

Private Member Functions

- void notifyDataBaseChange ()
- void writeToSD (String text)

Private Attributes

- final String TAG = "Social Proximity"
- DataBase database
- boolean debug = true
- ArrayList< DataBaseChangeListener > listeners = new ArrayList<DataBaseChangeListener> ()

6.10.1 Detailed Description

This class is responsible for computing the social weight among users.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 com.copelabs.socio.socialproximity.OnSocialWeightUpdate.OnSocialWeightUpdate (DataBase *database2*, SocialProximity *callback*)

This method is the constructor for OnSocialWeightUpdate.

Parameters

database2	The database to be used.
callback	The SocialProximity object to allow notification on social weight changes.

6.10.3 Member Function Documentation

6.10.3.1 void com.copelabs.socio.socialproximity.OnSocialWeightUpdate.computeSocialWeight (int *savDay,* int *currDay,* int *currTimeSlot,* long *currTime,* boolean *appR,* boolean *updOverDiffDays*)

This method computes the social weight towards all encountered nodes

Parameters

savDay	The day the app started running.
currDay	The current day when the app restarted.
currTimeSlot	The current time slot when the app restarted.
currTime	The current time when the app restarted.
appR	The flag that tells whether the app restarted.
updOverDiffDays	The flag that indicates the updates occur over different days.

6.10.3.2 int com.copelabs.socio.socialproximity.OnSocialWeightUpdate.getTimeSlot ( )

This method provides the current time slot.

Returns

currentTimeSlot The actual time slot.

6.10.3.3 void com.copelabs.socio.socialproximity.OnSocialWeightUpdate.notifyDataBaseChange ( ) [private]

This method notifies a database change to the listeners.

6.10.3.4 void com.copelabs.socio.socialproximity.OnSocialWeightUpdate.onReceive (Context *context,* Intent *intent*)

This method receives action concerning social weight updates.

Parameters

context	The context.
intent	The intent.

6.10.3.5 void com.copelabs.socio.socialproximity.OnSocialWeightUpdate.showDevicesOnDB ( )

This method shows the Bluetooth devices stored on the DB.

6.10.3.6 void com.copelabs.socio.socialproximity.OnSocialWeightUpdate.writeToSD (String *text*) [private]

Writes on a file for Bluetooth debugging

Parameters

text	The text to be printed.

6.10.4 Member Data Documentation

6.10.4.1 DataBase com.copelabs.socio.socialproximity.OnSocialWeightUpdate.database [private]

Sets the database used for social weight computation

6.10.4.2 int com.copelabs.socio.socialproximity.OnSocialWeightUpdate.day = 1 [static]

Sets the beginning of the day count used for social weight computation

6.10.4.3 boolean com.copelabs.socio.socialproximity.OnSocialWeightUpdate.debug = true [private]

Flag used for debugging purposes

6.10.4.4 ArrayList<DataBaseChangeListener> com.copelabs.socio.socialproximity.OnSocialWeightUpdate.listeners = new ArrayList<DataBaseChangeListener> () [private]

6.10.4.5 final String com.copelabs.socio.socialproximity.OnSocialWeightUpdate.NEW_HOUR [static]

Initial value:

```
=
            "android.intent.action.NEWHOUR"
```

Defines update action

6.10.4.6 final String com.copelabs.socio.socialproximity.OnSocialWeightUpdate.TAG = "Social Proximity" [private]

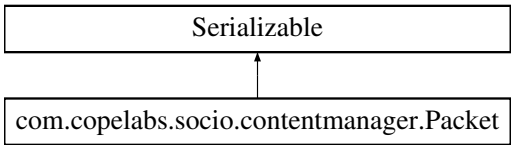
Tag used for debugging purposes

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

- src/com/copelabs/socio/socialproximity/OnSocialWeightUpdate.java

6.11 com.copelabs.socio.contentmanager.Packet Class Reference

Inheritance diagram for com.copelabs.socio.contentmanager.Packet:



Public Member Functions

- Packet ()
- Packet (String myBTMAC, String myBTName)
- String getIdSource ()
- String getIdDestination ()
- String getNameSource ()
- String getNameDestination ()
- String getApplication ()
- String getMessage ()
- long getTimestamp ()
- void setIdSource (String idSource)
- void setNameSource (String nameSource)
- void setIdDestination (String idDestination)
- void setNameDestination (String nameDestination)
- void setApplication (String application)
- void setAttributes (String idSource, String nameSource, String idDestination, String nameDestination, String application, String message, long timestamp)
- void setMessage (String msg)
- void setTimestamp (long ts)
- String getXmlEntry ()

Static Public Attributes

- static final String TAG_ID_SOURCE = "idSource"
- static final String TAG_NAME_SOURCE = "nameSource"
- static final String TAG_ID_DESTINATION = "idDestination"
- static final String TAG_NAME_DESTINATION = "nameDestination"
- static final String TAG_APPLICATION = "application"
- static final String TAG_MESSAGE = "message"
- static final String TAG_TIMESTAMP = "timestamp"

Private Attributes

- String idSource
- String nameSource
- String idDestination
- String nameDestination
- String application
- String message =""
- long timestamp

Static Private Attributes

- static final long serialVersionUID = 1L

6.11.1 Detailed Description

Parameters

idSource	the identifier of the owner's contact (BT MAC)
nameSource	the name of the Sender contact (BT Name)
idDestination	the identifier of the selected contact (BT MAC)
nameDestination	the name of the Receiver contact (BT Name)
message	the message content
timestamp	the timestamp for writing the message

6.11.2 Constructor & Destructor Documentation

6.11.2.1 com.copelabs.socio.contentmanager.Packet.Packet ( )

6.11.2.2 com.copelabs.socio.contentmanager.Packet.Packet (String *myBTMAC,* String *myBTName*)

6.11.3 Member Function Documentation

6.11.3.1 String com.copelabs.socio.contentmanager.Packet.getApplication ( )

6.11.3.2 String com.copelabs.socio.contentmanager.Packet.getIdDestination ( )

6.11.3.3 String com.copelabs.socio.contentmanager.Packet.getIdSource ( )

6.11.3.4 String com.copelabs.socio.contentmanager.Packet.getMessage ( )

6.11.3.5 String com.copelabs.socio.contentmanager.Packet.getNameDestination ( )

6.11.3.6 String com.copelabs.socio.contentmanager.Packet.getNameSource ( )

6.11.3.7 long com.copelabs.socio.contentmanager.Packet.getTimestamp ( )

6.11.3.8 String com.copelabs.socio.contentmanager.Packet.getXmlEntry ( )

Writes an OiMessage (List) in xml format

Returns

String an entry containing the xml format for an OiMessage

6.11.3.9 void com.copelabs.socio.contentmanager.Packet.setApplication (String *application*)

6.11.3.10 void com.copelabs.socio.contentmanager.Packet.setAttributes (String *idSource,* String *nameSource,* String *idDestination,* String *nameDestination,* String *application,* String *message,* long *timestamp*)

6.11.3.11 void com.copelabs.socio.contentmanager.Packet.setIdDestination (String *idDestination*)

6.11.3.12 void com.copelabs.socio.contentmanager.Packet.setIdSource (String *idSource*)

6.11.3.13 void com.copelabs.socio.contentmanager.Packet.setMessage (String *msg*)

6.11.3.14 void com.copelabs.socio.contentmanager.Packet.setNameDestination (String *nameDestination*)

6.11.3.15 void com.copelabs.socio.contentmanager.Packet.setNameSource (String *nameSource*)

6.11.3.16 void com.copelabs.socio.contentmanager.Packet.setTimestamp (long *ts*)

6.11.4 Member Data Documentation

6.11.4.1 String com.copelabs.socio.contentmanager.Packet.application [private]

6.11.4.2 String com.copelabs.socio.contentmanager.Packet.idDestination [private]

6.11.4.3 String com.copelabs.socio.contentmanager.Packet.idSource [private]

6.11.4.4 String com.copelabs.socio.contentmanager.Packet.message ="" [private]

6.11.4.5 String com.copelabs.socio.contentmanager.Packet.nameDestination [private]

6.11.4.6 String com.copelabs.socio.contentmanager.Packet.nameSource [private]

6.11.4.7 final long com.copelabs.socio.contentmanager.Packet.serialVersionUID = 1L [static], [private]

6.11.4.8 final String com.copelabs.socio.contentmanager.Packet.TAG_APPLICATION = "application" [static]

6.11.4.9 final String com.copelabs.socio.contentmanager.Packet.TAG_ID_DESTINATION = "idDestination" [static]

6.11.4.10 final String com.copelabs.socio.contentmanager.Packet.TAG_ID_SOURCE = "idSource" [static]

6.11.4.11 final String com.copelabs.socio.contentmanager.Packet.TAG_MESSAGE = "message" [static]

6.11.4.12 final String com.copelabs.socio.contentmanager.Packet.TAG_NAME_DESTINATION = "nameDestination" [static]

6.11.4.13 final String com.copelabs.socio.contentmanager.Packet.TAG_NAME_SOURCE = "nameSource" [static]

6.11.4.14 final String com.copelabs.socio.contentmanager.Packet.TAG_TIMESTAMP = "timestamp" [static]

6.11.4.15 long com.copelabs.socio.contentmanager.Packet.timestamp [private]

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

- src/com/copelabs/socio/contentmanager/Packet.java

6.12 com.copelabs.socio.router.Routing Class Reference

Public Member Functions

- Routing (Context context)
- List< UserDevice > getContactList ()
- Map< String, Integer > getSWList (ArrayList< String > mDevices)
- Map< String, Integer > getAllSWList ()
- List< String > getLocalInfo ()
- String getLocalMacAddress ()
- String getLocalName ()
- void stop ()
- void setRoutingListener (RoutingListener listener)
- void newContentAvailable (Packet mPacket)

Public Attributes

- ArrayList< RoutingListener > listeners = new ArrayList<RoutingListener> ()

Static Public Attributes

- static final String TAG ="Routing"

Private Member Functions

- String addColonMAC (String toConvert)
- void announceListSocialWeight (Map< String, Integer > mList)
- Map< String, String > convertMapValueDoubleToString (Map< String, Integer > map)
- void notifyPacketReceived (List< Packet > mListOfPackets)
- Map< String, List< Packet > > getListOfPackets (List< UserDevice > mListOfUserDevices)
- void notifyNewDeviceFound (UserDevice mUserDevice)
- void notifyDeviceDisappear (UserDevice mUserDevice)
- void notifyContactListUpdated (List< UserDevice > mList)
- void notifyError (int mError)

Private Attributes

- SocialProximity pSocialProximity
- WiFiDirect pWiFiDirect

6.12.1 Constructor & Destructor Documentation

6.12.1.1 com.copelabs.socio.router.Routing.Routing (Context *context*)

Routing constructor. It initializes the social proximity and WiFi direct modules and the respective listeners.

Parameters

context	Interface to global information about an application environment.

6.12.2 Member Function Documentation

6.12.2.1 String com.copelabs.socio.router.Routing.addColonMAC (String *toConvert*) [private]

Used to complete the mac address with colons

Parameters

toConvert	MAC address to convert

Returns

complete MAC address

6.12.2.2 void com.copelabs.socio.router.Routing.announceListSocialWeight (Map< String, Integer > *mList*) [private]

Used to update broadcast information at the interface layer.

Parameters

mList	- List of known contacts and their social weight.

6.12.2.3 Map<String, String> com.copelabs.socio.router.Routing.convertMapValueDoubleToString (Map< String, Integer > *map*) [private]

Convert Map<String, Double> to Map<String, String> needed by the WiFi Direct

Parameters

map	Map to be converted.

Returns

Map converted

6.12.2.4 Map<String, Integer> com.copelabs.socio.router.Routing.getAllSWList ( )

Provides a list of all encountered devices and social weight towards them

Returns

The list of all encountered devices and social weight towards them, or null

6.12.2.5 List<UserDevice> com.copelabs.socio.router.Routing.getContactList ( )

Provides a list of contact based on the information of devices encountered

Returns

The list of contacts or null

6.12.2.6 Map<String, List<Packet> > com.copelabs.socio.router.Routing.getListOfPackets (List< UserDevice > *mListOfUserDevices*) [private]

Provides a list of available packets to be sent to the given peers

Returns

List of Packets available to send or null if Content Manager is not available.

6.12.2.7 List<String> com.copelabs.socio.router.Routing.getLocalInfo ( )

Provides the info of local Bluetooth adapter

Returns

localMacAdd The local MAC Address or null

6.12.2.8 String com.copelabs.socio.router.Routing.getLocalMacAddress ( )

Provides the MAC Address of local Bluetooth adapter

Returns

BT MAC address

6.12.2.9 String com.copelabs.socio.router.Routing.getLocalName ( )

Provides the device name of local Bluetooth adapter

Returns

Device name

6.12.2.10 Map<String, Integer> com.copelabs.socio.router.Routing.getSWList (ArrayList< String > *mDevices*)

Provides a list of encountered devices and social weight towards them

Returns

The list of encountered devices and social weight towards them, or null

6.12.2.11 void com.copelabs.socio.router.Routing.newContentAvailable (Packet *mPacket*)

When new packets are received, a new SW list from neightbours is requested.

Parameters

mPacket	

6.12.2.12 void com.copelabs.socio.router.Routing.notifyContactListUpdated (List< UserDevice > *mList*) [private]

Provides a updated contact list.

Parameters

mList	Contact list

6.12.2.13 void com.copelabs.socio.router.Routing.notifyDeviceDisappear (UserDevice *mUserDevice*) [private]

Send a notification with the device info that is no longer available.

Parameters

mUserDevice	Device info

6.12.2.14 void com.copelabs.socio.router.Routing.notifyError (int *mError*) [private]

Notifies a error occurred inside the framework.

Parameters

mError	- Error code

6.12.2.15 void com.copelabs.socio.router.Routing.notifyNewDeviceFound (UserDevice *mUserDevice*) [private]

Provides the info about the device found.

Parameters

mUserDevice	Device info

6.12.2.16 void com.copelabs.socio.router.Routing.notifyPacketReceived (List< Packet > *mListOfPackets*) [private]

Notifies the ContentManager class that a new packet arrived.

Parameters

mListOfPackets	List of packets received

6.12.2.17 void com.copelabs.socio.router.Routing.setRoutingListener (RoutingListener *listener*)

Set the listener. Used by the content manager to receive info from the routing.

Parameters

listener	

6.12.2.18 void com.copelabs.socio.router.Routing.stop ( )

Stops the Routing module, including the WiFiDirect and SocialProximity modules.

6.12.3 Member Data Documentation

6.12.3.1 ArrayList<RoutingListener> com.copelabs.socio.router.Routing.listeners = new ArrayList<RoutingListener> ()

Used to send data to the Content Manager

6.12.3.2 SocialProximity com.copelabs.socio.router.Routing.pSocialProximity [private]

Social Proximity Module

6.12.3.3 WiFiDirect com.copelabs.socio.router.Routing.pWiFiDirect [private]

WiFi Direct Module

6.12.3.4 final String com.copelabs.socio.router.Routing.TAG ="Routing" [static]

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

- src/com/copelabs/socio/router/Routing.java

6.13 com.copelabs.socio.router.RoutingListener Interface Reference

Public Member Functions

- void onPacketReceived (List< Packet > mListOfPackets)
- Map< String, List< Packet > > getListOfPackets (List< UserDevice > mListOfUserDevices)
- void newDeviceFound (UserDevice mDevice)
- void deviceLost (UserDevice mDevice)
- void contactListUpdated (List< UserDevice > mListOfPackets)
- void error (int mError)

6.13.1 Member Function Documentation

6.13.1.1 void com.copelabs.socio.router.RoutingListener.contactListUpdated (List< UserDevice > *mListOfPackets*)

6.13.1.2 void com.copelabs.socio.router.RoutingListener.deviceLost (UserDevice *mDevice*)

6.13.1.3 void com.copelabs.socio.router.RoutingListener.error (int *mError*)

6.13.1.4 Map<String, List<Packet> > com.copelabs.socio.router.RoutingListener.getListOfPackets (List< UserDevice > *mListOfUserDevices*)

6.13.1.5 void com.copelabs.socio.router.RoutingListener.newDeviceFound (UserDevice *mDevice*)

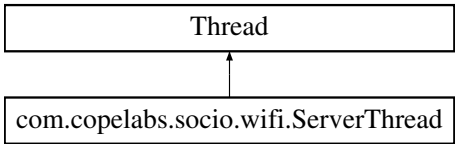
6.13.1.6 void com.copelabs.socio.router.RoutingListener.onPacketReceived (List< Packet > *mListOfPackets*)

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

- src/com/copelabs/socio/router/RoutingListener.java

6.14 com.copelabs.socio.wifi.ServerThread Class Reference

Inheritance diagram for com.copelabs.socio.wifi.ServerThread:



Public Member Functions

- ServerThread (Handler handler, Map< String, List< Packet >> mToSend) throws IOException
- void run ()

Static Public Attributes

- static final int SERVER_PORT = 4545

Private Attributes

- Handler handler
- Map< String, List< Packet > > mToSend

Static Private Attributes

- static final String TAG = "GroupOwnerSocketHandler"

6.14.1 Detailed Description

The implementation of a ServerSocket handler. This is used by the wifi p2p group owner.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 com.copelabs.socio.wifi.ServerThread.ServerThread (Handler *handler,* Map< String, List< Packet >> *mToSend*) throws IOException

ServerThread constructor

Parameters

handler	A Handler allows you to send and process Message.
mToSend	Content to be send

Exceptions

IOException	Signals a general, I/O-related error.

6.14.3 Member Function Documentation

6.14.3.1 void com.copelabs.socio.wifi.ServerThread.run ( )

6.14.4 Member Data Documentation

6.14.4.1 Handler com.copelabs.socio.wifi.ServerThread.handler [private]

A Handler allows you to send and process Message.

6.14.4.2 Map<String, List<Packet> > com.copelabs.socio.wifi.ServerThread.mToSend [private]

Content to be send

6.14.4.3 final int com.copelabs.socio.wifi.ServerThread.SERVER_PORT = 4545 [static]

Server port

6.14.4.4 final String com.copelabs.socio.wifi.ServerThread.TAG = "GroupOwnerSocketHandler" [static], [private]

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

- src/com/copelabs/socio/wifi/ServerThread.java

6.15 com.copelabs.socio.socialproximity.SocialProximity Class Reference

Classes

- class CustomComparator
- class **ServiceBTListener**

Public Member Functions

- SocialProximity (Context context)
- boolean isDeviceOnDB (String deviceAdd)
- int getTimeSlot ()
- List< UserDevice > getContactList ()
- Map< String, Integer > getSWList (ArrayList< String > mDevices)
- Map< String, Integer > getAllSWList ()
- List< String > getLocalInfo ()
- void stop ()
- void setSocialProximityListener (SocialProximityListener listener)
- void notifySWListUpdate ()
- void notifyNewDeviceEntry ()

Static Public Member Functions

- static void appRestartReset ()

Public Attributes

- ArrayList< SocialProximityListener > listeners = new ArrayList<SocialProximityListener> ()

Static Public Attributes

- static boolean appRestarted = false
- static final String DATABASE_CHANGE = "com.social.proximity.CHANGE"

Private Member Functions

- void initializeModules (Context context)
- void setNewHourAlarm ()
- void createPrefFile ()
- void notifyDataBaseChange ()

Static Private Member Functions

- static void writeToSD (String text)

Private Attributes

- BluetoothManager myBTManager
- ServiceBTListener btListener
- AlarmManager alarmMgr
- PendingIntent alarmIntent
- OnSocialWeightUpdate newHour
- Context context
- DataBase database

Static Private Attributes

- static final String TAG = "Social Proximity"
- static boolean debug = false

6.15.1 Detailed Description

This class is contains the core functionalities of the application. The BTManager provides all the information from Bluetooth adapter so this class can perform the social context analysis prior to storing the required information in the database.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 com.copelabs.socio.socialproximity.SocialProximity.SocialProximity (Context *context*)

This method is the constructor for SocialProximity.

Parameters

context	The context.

6.15.3 Member Function Documentation

6.15.3.1 static void com.copelabs.socio.socialproximity.SocialProximity.appRestartReset ( ) [static]

This method resets flag for when app was restarted

6.15.3.2 void com.copelabs.socio.socialproximity.SocialProximity.createPrefFile ( ) [private]

This method creates the preference file with information about the day and time slot when the app was started. If the file exists, updates it accordingly

6.15.3.3 Map<String, Integer> com.copelabs.socio.socialproximity.SocialProximity.getAllSWList ( )

This method provides a list of all social weights to send to neighbors.

Returns

listAllWeights The list of all computed social weights.

6.15.3.4 List<UserDevice> com.copelabs.socio.socialproximity.SocialProximity.getContactList ( )

This method provides a list of contact based on the information of devices encountered.

Returns

listOfContacts The list of contacts.

6.15.3.5 List<String> com.copelabs.socio.socialproximity.SocialProximity.getLocalInfo ( )

This method provides the name and the MAC Address of local Bluetooth adapter.

Returns

The local info or null

6.15.3.6 Map<String, Integer> com.copelabs.socio.socialproximity.SocialProximity.getSWList (ArrayList< String > *mDevices*)

This method provides a list of social weights for forwarding decisions.

Parameters

mDevices	The list of devices to get the SW.

Returns

swList The list of social weights.

6.15.3.7 int com.copelabs.socio.socialproximity.SocialProximity.getTimeSlot ( )

This method provides the current time slot.

Returns

currentTimeSlot The actual time slot.

6.15.3.8 void com.copelabs.socio.socialproximity.SocialProximity.initializeModules (Context *context*) [private]

This method initializes all modules used by Social Proximity. Each module is located in different packages.

Parameters

context	The context.

Creates a file for Bluetooth debugging

6.15.3.9 boolean com.copelabs.socio.socialproximity.SocialProximity.isDeviceOnDB (String *deviceAdd*)

This method checks whether the device is in DB

Parameters

deviceAdd	The device MAC address.

Returns

true if device is in DB, false otherwise

6.15.3.10 void com.copelabs.socio.socialproximity.SocialProximity.notifyDataBaseChange ( ) [private]

This method notifies a database change to the listeners.

6.15.3.11 void com.copelabs.socio.socialproximity.SocialProximity.notifyNewDeviceEntry ( )

This method notifies the Routing class of a contact list update.

6.15.3.12 void com.copelabs.socio.socialproximity.SocialProximity.notifySWListUpdate ( )

This method notifies the Routing class that social list has been updated.

6.15.3.13 void com.copelabs.socio.socialproximity.SocialProximity.setNewHourAlarm ( ) [private]

This method sets an alarm triggered every four minutes for social weight computation.

6.15.3.14 void com.copelabs.socio.socialproximity.SocialProximity.setSocialProximityListener (SocialProximityListener *listener*)

This method sets the listener that Routing will use to inform about update on the social list.

Parameters

listener	The listener to register

6.15.3.15 void com.copelabs.socio.socialproximity.SocialProximity.stop ( )

This method unregisters BroadcastReceiver, cancels the alarm, and closes the Bluetooth manager.

6.15.3.16 static void com.copelabs.socio.socialproximity.SocialProximity.writeToSD (String *text*) [static], [private]

Writes on a file for Bluetooth debugging

Parameters

text	The text to be printed

6.15.4 Member Data Documentation

6.15.4.1 PendingIntent com.copelabs.socio.socialproximity.SocialProximity.alarmIntent [private]

6.15.4.2 AlarmManager com.copelabs.socio.socialproximity.SocialProximity.alarmMgr [private]

6.15.4.3 boolean com.copelabs.socio.socialproximity.SocialProximity.appRestarted = false [static]

6.15.4.4 ServiceBTListener com.copelabs.socio.socialproximity.SocialProximity.btListener [private]

6.15.4.5 Context com.copelabs.socio.socialproximity.SocialProximity.context [private]

6.15.4.6 DataBase com.copelabs.socio.socialproximity.SocialProximity.database [private]

6.15.4.7 final String com.copelabs.socio.socialproximity.SocialProximity.DATABASE_CHANGE = "com.social.proximity.CHANGE" [static]

6.15.4.8 boolean com.copelabs.socio.socialproximity.SocialProximity.debug = false [static], [private]

6.15.4.9 ArrayList<SocialProximityListener> com.copelabs.socio.socialproximity.SocialProximity.listeners = new ArrayList<SocialProximityListener> ()

6.15.4.10 BluetoothManager com.copelabs.socio.socialproximity.SocialProximity.myBTManager [private]

6.15.4.11 OnSocialWeightUpdate com.copelabs.socio.socialproximity.SocialProximity.newHour [private]

6.15.4.12 final String com.copelabs.socio.socialproximity.SocialProximity.TAG = "Social Proximity" [static], [private]

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

- src/com/copelabs/socio/socialproximity/SocialProximity.java

6.16 com.copelabs.socio.socialproximity.SocialProximityListener Interface Reference

Public Member Functions

- void notifySWListUpdate ()
- void notifyNewDeviceEntry ()

6.16.1 Member Function Documentation

6.16.1.1 void com.copelabs.socio.socialproximity.SocialProximityListener.notifyNewDeviceEntry ( )

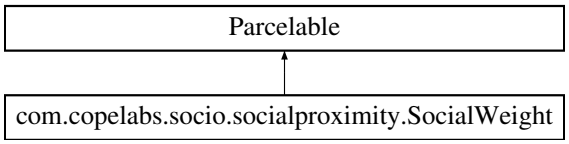
6.16.1.2 void com.copelabs.socio.socialproximity.SocialProximityListener.notifySWListUpdate ( )

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

- src/com/copelabs/socio/socialproximity/SocialProximityListener.java

6.17 com.copelabs.socio.socialproximity.SocialWeight Class Reference

Inheritance diagram for com.copelabs.socio.socialproximity.SocialWeight:



Public Member Functions

- SocialWeight (String mMacAddress, String mDeviceName, double mSocialWeight)
- void setMacAddress (String mMacAddress)
- String getMacAddress ()
- void setDeviceName (String mDeviceName)
- String getDeviceName ()
- void setSocialWeight (long mSocialWeight)
- double getSocialWeight ()
- int describeContents ()
- void writeToParcel (Parcel dest, int flags)

Public Attributes

- final Creator<?> CREATOR

Static Public Attributes

- static final String socialWeight_key = "socialReceiver"

Private Member Functions

- SocialWeight (Parcel source)

Private Attributes

- String mMacAddress
- String mDeviceName
- double mSocialWeight

6.17.1 Detailed Description

This class holds information about a device: name, MAC address, and social weight towards it.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 com.copelabs.socio.socialproximity.SocialWeight.SocialWeight (String *mMacAddress,* String *mDeviceName,* double *mSocialWeight*)

This method is the constructor for SocialWeight.

Parameters

mMacAddress	The MAC address of the device.
mDeviceName	The name of the device.
mSocialWeight	The social weight towards the device.

6.17.2.2 com.copelabs.socio.socialproximity.SocialWeight.SocialWeight (Parcel *source*) [private]

This method is the constructor for SocialWeight.

Parameters

source	The Parcel source from which information can be extracted.

6.17.3 Member Function Documentation

6.17.3.1 int com.copelabs.socio.socialproximity.SocialWeight.describeContents ( )

6.17.3.2 String com.copelabs.socio.socialproximity.SocialWeight.getDeviceName ( )

This method gets the device's name.

6.17.3.3 String com.copelabs.socio.socialproximity.SocialWeight.getMacAddress ( )

This method gets the device's MAC address.

6.17.3.4 double com.copelabs.socio.socialproximity.SocialWeight.getSocialWeight ( )

This method gets the device's social weight.

6.17.3.5 void com.copelabs.socio.socialproximity.SocialWeight.setDeviceName (String *mDeviceName*)

This method sets the device's name.

Parameters

mDeviceName	The device's name.

6.17.3.6 void com.copelabs.socio.socialproximity.SocialWeight.setMacAddress (String *mMacAddress*)

This method sets the device's MAC address.

Parameters

mMacAddress	The device's MAC address.

6.17.3.7 void com.copelabs.socio.socialproximity.SocialWeight.setSocialWeight (long *mSocialWeight*)

This method sets the device's social weight.

Parameters

mSocialWeight	The device's social weight.

6.17.3.8 void com.copelabs.socio.socialproximity.SocialWeight.writeToParcel (Parcel *dest*, int *flags*)

6.17.4 Member Data Documentation

6.17.4.1 final Creator<?> com.copelabs.socio.socialproximity.SocialWeight.CREATOR

Initial value:

```
= new Creator<Object>() {
        @Override
        public SocialWeight createFromParcel(Parcel in) {
            return new SocialWeight(in);
        }
        @Override
        public SocialWeight[] newArray(int size) {
            return new SocialWeight[size];
        }
    }
```

6.17.4.2 String com.copelabs.socio.socialproximity.SocialWeight.mDeviceName [private]

Device name

6.17.4.3 String com.copelabs.socio.socialproximity.SocialWeight.mMacAddress [private]

MAC Address

6.17.4.4 double com.copelabs.socio.socialproximity.SocialWeight.mSocialWeight [private]

Social Weight

6.17.4.5 final String com.copelabs.socio.socialproximity.SocialWeight.socialWeight_key = "socialReceiver" [static]

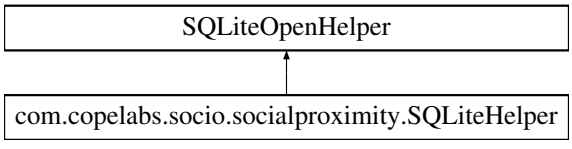
index Key

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

- src/com/copelabs/socio/socialproximity/SocialWeight.java

6.18 com.copelabs.socio.socialproximity.SQLiteHelper Class Reference

Inheritance diagram for com.copelabs.socio.socialproximity.SQLiteHelper:



Public Member Functions

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

Static Public Attributes

- static final String TABLE_BTDEVICE = "btdevices"
- static final String TABLE_BTDEVICEENCOUNTERDURATION = "btdevice_encounterduration"
- static final String TABLE_BTDEVICEAVERAGEENCOUNTERDURATION = "btdevice_averageencounterduration"
- static final String TABLE_BTDEVICESOCIALWEIGHT = "btdevice_socialweight"
- static final String COLUMN_ID = "_id"
- static final String COLUMN_BTDEV_MAC_ADDRESS = "devBtMacAdd"
- static final String COLUMN_BTDEV_NAME = "devName"
- static final String COLUMN_BTDEV_ENCOUNTERSTART = "devEncounterStart"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT1 = "devEncounterDuration_slot1"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT2 = "devEncounterDuration_slot2"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT3 = "devEncounterDuration_slot3"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT4 = "devEncounterDuration_slot4"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT5 = "devEncounterDuration_slot5"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT6 = "devEncounterDuration_slot6"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT7 = "devEncounterDuration_slot7"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT8 = "devEncounterDuration_slot8"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT9 = "devEncounterDuration_slot9"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT10 = "devEncounterDuration_slot10"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT11 = "devEncounterDuration_slot11"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT12 = "devEncounterDuration_slot12"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT13 = "devEncounterDuration_slot13"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT14 = "devEncounterDuration_slot14"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT15 = "devEncounterDuration_slot15"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT16 = "devEncounterDuration_slot16"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT17 = "devEncounterDuration_slot17"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT18 = "devEncounterDuration_slot18"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT19 = "devEncounterDuration_slot19"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT20 = "devEncounterDuration_slot20"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT21 = "devEncounterDuration_slot21"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT22 = "devEncounterDuration_slot22"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT23 = "devEncounterDuration_slot23"
- static final String COLUMN_BTDEV_ENCOUNTERDURATION_SLOT24 = "devEncounterDuration_slot24"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT1 = "devAvgEncounterDuration↩ _slot1"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT2 = "devAvgEncounterDuration↩ _slot2"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT3 = "devAvgEncounterDuration↩ _slot3"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT4 = "devAvgEncounterDuration↩ _slot4"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT5 = "devAvgEncounterDuration↩ _slot5"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT6 = "devAvgEncounterDuration↩ _slot6"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT7 = "devAvgEncounterDuration↩ _slot7"

- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT8 = "devAvgEncounterDuration_slot8"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT9 = "devAvgEncounterDuration_slot9"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT10 = "devAvgEncounterDuration_slot10"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT11 = "devAvgEncounterDuration_slot11"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT12 = "devAvgEncounterDuration_slot12"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT13 = "devAvgEncounterDuration_slot13"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT14 = "devAvgEncounterDuration_slot14"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT15 = "devAvgEncounterDuration_slot15"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT16 = "devAvgEncounterDuration_slot16"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT17 = "devAvgEncounterDuration_slot17"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT18 = "devAvgEncounterDuration_slot18"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT19 = "devAvgEncounterDuration_slot19"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT20 = "devAvgEncounterDuration_slot20"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT21 = "devAvgEncounterDuration_slot21"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT22 = "devAvgEncounterDuration_slot22"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT23 = "devAvgEncounterDuration_slot23"
- static final String COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT24 = "devAvgEncounterDuration_slot24"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT1 = "devSocialWeight_slot1"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT2 = "devSocialWeight_slot2"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT3 = "devSocialWeight_slot3"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT4 = "devSocialWeight_slot4"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT5 = "devSocialWeight_slot5"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT6 = "devSocialWeight_slot6"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT7 = "devSocialWeight_slot7"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT8 = "devSocialWeight_slot8"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT9 = "devSocialWeight_slot9"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT10 = "devSocialWeight_slot10"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT11 = "devSocialWeight_slot11"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT12 = "devSocialWeight_slot12"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT13 = "devSocialWeight_slot13"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT14 = "devSocialWeight_slot14"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT15 = "devSocialWeight_slot15"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT16 = "devSocialWeight_slot16"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT17 = "devSocialWeight_slot17"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT18 = "devSocialWeight_slot18"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT19 = "devSocialWeight_slot19"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT20 = "devSocialWeight_slot20"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT21 = "devSocialWeight_slot21"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT22 = "devSocialWeight_slot22"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT23 = "devSocialWeight_slot23"
- static final String COLUMN_BTDEV_SOCIALWEIGHT_SLOT24 = "devSocialWeight_slot24"

Static Private Attributes

- static final String DATABASE_NAME = "database"
- static final int DATABASE_VERSION = 2
- static final String CREATE_BTDEVICE_TABLE
- static final String CREATE_BTDEVICEENCOUNTERDURATION_TABLE
- static final String CREATE_BTDEVICEAVERAGEENCOUNTERDURATION_TABLE
- static final String CREATE_BTDEVICESOCIALWEIGHT_TABLE

6.18.1 Detailed Description

This class defines the tables and their attributes used for social weight computation.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 com.copelabs.socio.socialproximity.SQLiteHelper.SQLiteHelper (Context *context*)

This method is the constructor for SQLiteHelper.

Parameters

context	The context.

6.18.3 Member Function Documentation

6.18.3.1 void com.copelabs.socio.socialproximity.SQLiteHelper.onCreate (SQLiteDatabase *dataBase*)

6.18.3.2 void com.copelabs.socio.socialproximity.SQLiteHelper.onUpgrade (SQLiteDatabase *dataBase,* int *oldVersion,* int *newVersion*)

6.18.4 Member Data Documentation

6.18.4.1 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT1 = "devAvgEncounterDuration_slot1" [static]

6.18.4.2 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT10 = "devAvgEncounterDuration_slot10" [static]

6.18.4.3 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT11 = "devAvgEncounterDuration_slot11" [static]

6.18.4.4 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT12 = "devAvgEncounterDuration_slot12" [static]

6.18.4.5 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT13 = "devAvgEncounterDuration_slot13" [static]

6.18.4.6 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT14 = "devAvgEncounterDuration_slot14" [static]

6.18.4.7 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT15 = "devAvgEncounterDuration_slot15" [static]

6.18.4.8 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT16 = "devAvgEncounterDuration_slot16" [static]

6.18.4.9 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT17 = "devAvgEncounterDuration_slot17" [static]

6.18.4.10 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT18 = "devAvgEncounterDuration_slot18" [static]

6.18.4.11 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT19 = "devAvgEncounterDuration_slot19" [static]

6.18.4.12 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT2 = "devAvgEncounterDuration_slot2" [static]

6.18.4.13 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT20 = "devAvgEncounterDuration_slot20" [static]

6.18.4.14 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT21 = "devAvgEncounterDuration_slot21" [static]

6.18.4.15 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT22 = "devAvgEncounterDuration_slot22" [static]

6.18.4.16 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT23 = "devAvgEncounterDuration_slot23" [static]

6.18.4.17 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT24 = "devAvgEncounterDuration_slot24" [static]

6.18.4.18 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT3 = "devAvgEncounterDuration_slot3" [static]

6.18.4.19 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT4 = "devAvgEncounterDuration_slot4" [static]

6.18.4.20 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT5 = "devAvgEncounterDuration_slot5" [static]

6.18.4.21 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT6 = "devAvgEncounterDuration_slot6" [static]

6.18.4.22 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT7 = "devAvgEncounterDuration_slot7" [static]

6.18.4.23 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT8 = "devAvgEncounterDuration_slot8" [static]

6.18.4.24 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT9 = "devAvgEncounterDuration_slot9" [static]

6.18.4.25 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT1 = "devEncounterDuration_slot1" [static]

6.18.4.26 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT10 = "devEncounterDuration_slot10" [static]

6.18.4.27 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT11 = "devEncounterDuration_slot11" [static]

6.18.4.28 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT12 = "devEncounterDuration_slot12" [static]

6.18.4.29 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT13 = "devEncounterDuration_slot13" [static]

6.18.4.30 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT14 = "devEncounterDuration_slot14" [static]

6.18.4.31 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT15 = "devEncounterDuration_slot15" [static]

6.18.4.32 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT16 = "devEncounterDuration_slot16" [static]

6.18.4.33 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT17 = "devEncounterDuration_slot17" [static]

6.18.4.34 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT18 = "devEncounterDuration_slot18" [static]

6.18.4.35 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT19 = "devEncounterDuration_slot19" [static]

6.18.4.36 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT2 = "devEncounterDuration_slot2" [static]

6.18.4.37 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT20 = "devEncounterDuration_slot20" [static]

6.18.4.38 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT21 = "devEncounterDuration_slot21" [static]

6.18.4.39 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT22 = "devEncounterDuration_slot22" [static]

6.18.4.40 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT23 = "devEncounterDuration_slot23" [static]

6.18.4.41 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT24 = "devEncounterDuration_slot24" [static]

6.18.4.42 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT3 = "devEncounterDuration_slot3" [static]

6.18.4.43 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT4 = "devEncounterDuration_slot4" [static]

6.18.4.44 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT5 = "devEncounterDuration_slot5" [static]

6.18.4.45 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT6 = "devEncounterDuration_slot6" [static]

6.18.4.46 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT7 = "devEncounterDuration_slot7" [static]

6.18.4.47 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT8 = "devEncounterDuration_slot8" [static]

6.18.4.48 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERDURATION_SLOT9 = "devEncounterDuration_slot9" [static]

6.18.4.49 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_ENCOUNTERSTART = "devEncounterStart" [static]

6.18.4.50 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_MAC_ADDRESS = "devBtMacAdd" [static]

6.18.4.51 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_NAME = "devName" [static]

6.18.4.52 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT1 = "devSocialWeight_slot1" [static]

6.18.4.53 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT10 = "devSocialWeight_slot10" [static]

6.18.4.54 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT11 = "devSocialWeight_slot11" [static]

6.18.4.55 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT12 = "devSocialWeight_slot12" [static]

6.18.4.56 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT13 = "devSocialWeight_slot13" [static]

6.18.4.57 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT14 = "devSocialWeight_slot14" [static]

6.18.4.58 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT15 = "devSocialWeight_slot15" [static]

6.18.4.59 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT16 = "devSocialWeight_slot16" [static]

6.18.4.60 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT17 = "devSocialWeight_slot17" [static]

6.18.4.61 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT18 = "devSocialWeight_slot18" [static]

6.18.4.62 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT19 = "devSocialWeight_slot19" [static]

6.18.4.63 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT2 = "devSocialWeight_slot2" [static]

6.18.4.64 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT20 = "devSocialWeight_slot20" [static]

6.18.4.65 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT21 = "devSocialWeight_slot21" [static]

6.18.4.66 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT22 = "devSocialWeight_slot22" [static]

6.18.4.67 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT23 = "devSocialWeight_slot23" [static]

6.18.4.68 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT24 = "devSocialWeight_slot24" [static]

6.18.4.69 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT3 = "devSocialWeight_slot3" [static]

6.18.4.70 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT4 = "devSocialWeight_slot4" [static]

6.18.4.71 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT5 = "devSocialWeight_slot5" [static]

6.18.4.72 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT6 = "devSocialWeight_slot6" [static]

6.18.4.73 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT7 = "devSocialWeight_slot7" [static]

6.18.4.74 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT8 = "devSocialWeight_slot8" [static]

6.18.4.75 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_BTDEV_SOCIALWEIGHT_SLOT9 = "devSocialWeight_slot9" [static]

6.18.4.76 final String com.copelabs.socio.socialproximity.SQLiteHelper.COLUMN_ID = "_id" [static]

6.18.4.77 final String com.copelabs.socio.socialproximity.SQLiteHelper.CREATE_BTDEVICE_TABLE [static], [private]

Initial value:

```
= "create table "
                + TABLE_BTDEVICE + "("
                + COLUMN_ID + " integer primary key autoincrement, "
                + COLUMN_BTDEV_MAC_ADDRESS + " text not null unique, "
                + COLUMN_BTDEV_NAME + " text not null, "
                + COLUMN_BTDEV_ENCOUNTERSTART + " long not null"
                + ");"
```

6.18.4.78 final String com.copelabs.socio.socialproximity.SQLiteHelper.CREATE_BTDEVICEAVERAGEENCOUNTERDURATIO↩N_TABLE [static], [private]

Initial value:

```
= "create table "
                + TABLE_BTDEVICEAVERAGEENCOUNTERDURATION + "("
                + COLUMN_ID + " integer primary key autoincrement, "
                + COLUMN_BTDEV_MAC_ADDRESS + " text not null unique, "
                + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT1 + " double
        not null, "
                + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT2 + " double
        not null, "
                + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT3 + " double
        not null, "
                + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT4 + " double
        not null, "
```

```
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT5 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT6 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT7 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT8 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT9 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT10 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT11 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT12 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT13 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT14 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT15 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT16 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT17 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT18 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT19 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT20 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT21 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT22 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT23 + " double
        not null, "
            + COLUMN_BTDEV_AVGENCOUNTERDURATION_SLOT24 + " double
        not null"
            + ");"
```

6.18.4.79 final String com.copelabs.socio.socialproximity.SQLiteHelper.CREATE_BTDEVICEENCOUNTERDURATION_TABLE [static],[private]

Initial value:

```
= "create table "
            + TABLE_BTDEVICEENCOUNTERDURATION + "("
            + COLUMN_ID + " integer primary key autoincrement, "
            + COLUMN_BTDEV_MAC_ADDRESS + " text not null unique, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT1 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT2 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT3 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT4 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT5 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT6 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT7 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT8 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT9 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT10 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT11 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT12 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT13 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT14 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT15 + " double not
        null, "
            + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT16 + " double not
```

```
        null, "
                + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT17 + " double not
        null, "
                + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT18 + " double not
        null, "
                + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT19 + " double not
        null, "
                + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT20 + " double not
        null, "
                + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT21 + " double not
        null, "
                + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT22 + " double not
        null, "
                + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT23 + " double not
        null, "
                + COLUMN_BTDEV_ENCOUNTERDURATION_SLOT24 + " double not
        null"
                + ");"
```

6.18.4.80 final String com.copelabs.socio.socialproximity.SQLiteHelper.CREATE_BTDEVICESOCIALWEIGHT_TABLE [static], [private]

Initial value:

```
= "create table "
                + TABLE_BTDEVICESOCIALWEIGHT + "("
                + COLUMN_ID + " integer primary key autoincrement, "
                + COLUMN_BTDEV_MAC_ADDRESS + " text not null unique, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT1 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT2 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT3 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT4 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT5 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT6 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT7 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT8 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT9 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT10 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT11 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT12 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT13 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT14 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT15 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT16 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT17 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT18 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT19 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT20 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT21 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT22 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT23 + " double not null, "
                + COLUMN_BTDEV_SOCIALWEIGHT_SLOT24 + " double not null"
                + ");"
```

6.18.4.81 final String com.copelabs.socio.socialproximity.SQLiteHelper.DATABASE_NAME = "database" [static], [private]

6.18.4.82 final int com.copelabs.socio.socialproximity.SQLiteHelper.DATABASE_VERSION = 2 [static], [private]

6.18.4.83 final String com.copelabs.socio.socialproximity.SQLiteHelper.TABLE_BTDEVICE = "btdevices" [static]

6.18.4.84 final String com.copelabs.socio.socialproximity.SQLiteHelper.TABLE_BTDEVICEAVERAGEENCOUNTERDURATION = "btdevice_averageencounterduration" [static]

6.18.4.85 final String com.copelabs.socio.socialproximity.SQLiteHelper.TABLE_BTDEVICEENCOUNTERDURATION = "btdevice_encounterduration" [static]

6.18.4.86 final String com.copelabs.socio.socialproximity.SQLiteHelper.TABLE_BTDEVICESOCIALWEIGHT = "btdevice_socialweight" [static]

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

- src/com/copelabs/socio/socialproximity/SQLiteHelper.java

6.19 com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration Class Reference

Public Member Functions

- String getDevAdd ()
- double getAverageEncounterDuration (int timeSlot)
- void setDevAdd (String DevAdd)
- void setAverageEncounterDuration (int timeSlot, double avgEncounterDuration)
- UserDevAverageEncounterDuration ()

Private Attributes

- String deviceAdd
- double[ ] averageEncounterDuration = new double [24]

6.19.1 Detailed Description

This class holds the average duration of encounter between a peer and the user device in specific time slots.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration.UserDevAverageEncounterDuration ( )

BT User Device Average Encounter Duration Constructor

6.19.3 Member Function Documentation

6.19.3.1 double com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration.getAverageEncounterDuration (int *timeSlot*)

This method gets the average encounter duration that the BT device is within communication range.

Parameters

timeSlot	The specific time slot.

Returns

The average encounter duration in the given slot.

6.19.3.2 String com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration.getDevAdd ( )

This method gets the MAC address of this Bluetooth device.

Returns

deviceAdd The device address to be used as key.

6.19.3.3 void com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration.setAverageEncounterDuration (int *timeSlot,* double *avgEncounterDuration*)

This method sets the average duration that the BT device is within communication range.

Parameters

timeSlot	The specific time slot.
avgEncounter↩ Duration	The average duration of encounter.

6.19.3.4 void com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration.setDevAdd (String *DevAdd*)

This method sets the ID of this Bluetooth device.

Parameters

DevAdd	The MAC address of device to set.

6.19.4 Member Data Documentation

6.19.4.1 double [ ] com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration.averageEncounterDuration = new double [24] [private]

6.19.4.2 String com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration.deviceAdd [private]

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

- src/com/copelabs/socio/socialproximity/UserDevAverageEncounterDuration.java

6.20 com.copelabs.socio.socialproximity.UserDevEncounterDuration Class Reference

Public Member Functions

- String getDevAdd ()
- double getEncounterDuration (int timeSlot)
- void setDevAdd (String DevAdd)
- void setEncounterDuration (int timeSlot, double encounterDuration)
- UserDevEncounterDuration ()

Private Attributes

- String deviceAdd
- double[ ] encounterDuration = new double [24]

6.20.1 Detailed Description

This class holds the encounter duration between a peer and the user device in specific time slots.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 com.copelabs.socio.socialproximity.UserDevEncounterDuration.UserDevEncounterDuration ( )

BT User Device Encounter Duration Constructor

6.20.3 Member Function Documentation

6.20.3.1 String com.copelabs.socio.socialproximity.UserDevEncounterDuration.getDevAdd ( )

This method gets the MAC address of this Bluetooth device.

Returns

deviceAdd The device address to be used as key.

6.20.3.2 double com.copelabs.socio.socialproximity.UserDevEncounterDuration.getEncounterDuration (int *timeSlot*)

This method gets the duration that the BT device is within communication range.

Parameters

timeSlot	The specific time slot.

Returns

The encounter duration in the given slot.

6.20.3.3 void com.copelabs.socio.socialproximity.UserDevEncounterDuration.setDevAdd (String *DevAdd*)

This method sets the ID of this Bluetooth device.

Parameters

DevAdd	The MAC address of device to set.

6.20.3.4 void com.copelabs.socio.socialproximity.UserDevEncounterDuration.setEncounterDuration (int *timeSlot,* double *encounterDuration*)

This method sets the duration that the BT device is within communication range.

Parameters

timeSlot	The specific time slot.
encounter↩ Duration	The duration of encounter.

6.20.4 Member Data Documentation

6.20.4.1 String com.copelabs.socio.socialproximity.UserDevEncounterDuration.deviceAdd [private]

6.20.4.2 double [ ] com.copelabs.socio.socialproximity.UserDevEncounterDuration.encounterDuration = new double [24] [private]

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

- src/com/copelabs/socio/socialproximity/UserDevEncounterDuration.java

6.21 com.copelabs.socio.socialproximity.UserDeviceInfo Class Reference

Public Member Functions

- String getDevAdd ()
- String getDevName ()
- long getEncounterStart ()
- void setDevAdd (String DevAdd)
- void setDevName (String DevName)
- void setEncounterTime (long time)
- UserDeviceInfo ()

Private Attributes

- String deviceAdd
- String deviceName
- long encounterTime

6.21.1 Detailed Description

This class represents a user device found by means of Bluetooth.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 com.copelabs.socio.socialproximity.UserDeviceInfo.UserDeviceInfo ( )

Bluetooth User Device Constructor

6.21.3 Member Function Documentation

6.21.3.1 String com.copelabs.socio.socialproximity.UserDeviceInfo.getDevAdd ( )

This method gets the MAC address of this Bluetooth device.

Returns

deviceAdd The device address.

6.21.3.2 String com.copelabs.socio.socialproximity.UserDeviceInfo.getDevName ( )

This method gets the name of this Bluetooth device.

Returns

deviceName The device name.

6.21.3.3 long com.copelabs.socio.socialproximity.UserDeviceInfo.getEncounterStart ( )

This method gets the time that the BT device is is first found.

Returns

The encounter time.

6.21.3.4 void com.copelabs.socio.socialproximity.UserDeviceInfo.setDevAdd (String *DevAdd*)

This method sets the ID of this Bluetooth device.

Parameters

DevAdd	The MAC address of device to set.

6.21.3.5 void com.copelabs.socio.socialproximity.UserDeviceInfo.setDevName (String *DevName*)

This method sets the name of this Bluetooth device.

Parameters

DevName	The name of device to set.

6.21.3.6 void com.copelabs.socio.socialproximity.UserDeviceInfo.setEncounterTime (long *time*)

This method sets the time that the BT device is first found.

Parameters

time	The time.

6.21.4 Member Data Documentation

6.21.4.1 String com.copelabs.socio.socialproximity.UserDeviceInfo.deviceAdd [private]

6.21.4.2 String com.copelabs.socio.socialproximity.UserDeviceInfo.deviceName [private]

6.21.4.3 long com.copelabs.socio.socialproximity.UserDeviceInfo.encounterTime [private]

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

- src/com/copelabs/socio/socialproximity/UserDeviceInfo.java

6.22 com.copelabs.socio.socialproximity.UserDevSocialWeight Class Reference

Public Member Functions

- String getDevAdd ()
- double getSocialWeight (int timeSlot)
- void setDevAdd (String DevAdd)
- void setSocialWeight (int timeSlot, double socialWeight)
- UserDevSocialWeight ()

Private Attributes

- String deviceAdd
- double[ ] socialWeight = new double [24]

6.22.1 Detailed Description

This class holds the social weight between a peer and the user device in specific time slots.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 com.copelabs.socio.socialproximity.UserDevSocialWeight.UserDevSocialWeight ( )

BT User Device Social Weight Constructor

6.22.3 Member Function Documentation

6.22.3.1 String com.copelabs.socio.socialproximity.UserDevSocialWeight.getDevAdd ( )

This method gets the MAC address of this Bluetooth device.

Returns

deviceAdd The device address to be used as key.

6.22.3.2 double com.copelabs.socio.socialproximity.UserDevSocialWeight.getSocialWeight (int *timeSlot*)

This method gets the social weight towards a BT device.

Parameters

timeSlot	The specific time slot.

Returns

The social weight in the given slot.

6.22.3.3 void com.copelabs.socio.socialproximity.UserDevSocialWeight.setDevAdd (String *DevAdd*)

This method sets the ID of this Bluetooth device.

Parameters

DevAdd	The MAC address of device to set.

6.22.3.4 void com.copelabs.socio.socialproximity.UserDevSocialWeight.setSocialWeight (int *timeSlot,* double *socialWeight*)

This method sets the social weight towards a BT device.

Parameters

timeSlot	The specific time slot.
socialWeight	The social weight.

6.22.4 Member Data Documentation

6.22.4.1 String com.copelabs.socio.socialproximity.UserDevSocialWeight.deviceAdd [private]

6.22.4.2 double [ ] com.copelabs.socio.socialproximity.UserDevSocialWeight.socialWeight = new double [24] [private]

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

- src/com/copelabs/socio/socialproximity/UserDevSocialWeight.java

6.23 com.copelabs.socio.wifi.WiFiDirect Class Reference

Public Member Functions

- WiFiDirect (Context mContext)
- void start (String mMAC, Map< String, String > mListOfSocialWeight)
- void stop ()
- void setWiFiDirectListener (WiFiDirectListener listener)
- String getWiFiDirectMacAdress ()
- String getWiFiMacAddress ()
- boolean sendPackets (WiFiDirectDevice mDevice, ArrayList< Packet > mPackets)
- void updateAnnounce (String mMAC, Map< String, String > mListOfSocialWeight)
- void resetAvailableDevices ()

Public Attributes

- ArrayList< WiFiDirectListener > listeners = new ArrayList<WiFiDirectListener> ()

Static Public Attributes

- static final String TAG = "wifidirect"

Private Attributes

- WiFiDirectUtils mWiFiDirectUtils

6.23.1 Constructor & Destructor Documentation

6.23.1.1 com.copelabs.socio.wifi.WiFiDirect.WiFiDirect (Context *mContext*)

WiFiDirect constructor

Parameters

mContext	Interface to global information about an application environment.

6.23.2 Member Function Documentation

6.23.2.1 String com.copelabs.socio.wifi.WiFiDirect.getWiFiDirectMacAdress ( )

Get the MAC address of WiFi P2P interface. Only returns a MAC address if this device is a group owner.

Returns

WiFi P2P interface MAC address

6.23.2.2 String com.copelabs.socio.wifi.WiFiDirect.getWiFiMacAddress ( )

Get the MAC address of WiFi interface.

Returns

WiFi MAC address

6.23.2.3 void com.copelabs.socio.wifi.WiFiDirect.resetAvailableDevices ( )

Restarts WiFi Direct.

6.23.2.4 boolean com.copelabs.socio.wifi.WiFiDirect.sendPackets (WiFiDirectDevice *mDevice,* ArrayList< Packet > *mPackets*)

Used by the above modules to send a packet through WiFi P2P.

Parameters

mDevice	Destination device
mPackets	List of Packets to send

Returns

Always true.

6.23.2.5 void com.copelabs.socio.wifi.WiFiDirect.setWiFiDirectListener (WiFiDirectListener *listener*)

Listener that WiFi package will use to inform register packages about a new packet received or when new devices were found. Check class WiFiDirectListener to see the available outputs.

Parameters

listener	listener to register

6.23.2.6 void com.copelabs.socio.wifi.WiFiDirect.start (String *mMAC,* Map< String, String > *mListOfSocialWeight*)

It starts the WiFi Direct module. This function starts the list of social weight announce.

Parameters

mMAC	Local BT MAC address
mListOfSocial↩ Weight	List of known devices and their social weights.

6.23.2.7 void com.copelabs.socio.wifi.WiFiDirect.stop ( )

It stops the WiFiDirect module, including the announce of the social weights.

6.23.2.8 void com.copelabs.socio.wifi.WiFiDirect.updateAnnounce (String *mMAC,* Map< String, String > *mListOfSocialWeight*)

To update the info announced by the WiFi P2P (Bonjour service)

Parameters

mMAC	- Local MAC Address
mListOfSocial↩ Weight	- List of known contacts and their social weights

6.23.3 Member Data Documentation

6.23.3.1 ArrayList<WiFiDirectListener> com.copelabs.socio.wifi.WiFiDirect.listeners = new ArrayList<WiFiDirectListener> ()

List of listener that receive info from this module.

6.23.3.2 WiFiDirectUtils com.copelabs.socio.wifi.WiFiDirect.mWiFiDirectUtils [private]

WiFi Direct core functions

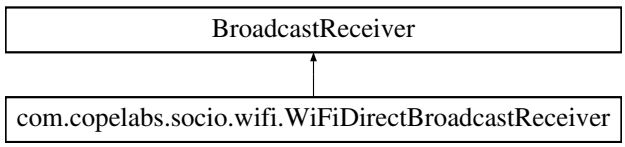
6.23.3.3 final String com.copelabs.socio.wifi.WiFiDirect.TAG = "wifidirect" [static]

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

- src/com/copelabs/socio/wifi/WiFiDirect.java

6.24 com.copelabs.socio.wifi.WiFiDirectBroadcastReceiver Class Reference

Inheritance diagram for com.copelabs.socio.wifi.WiFiDirectBroadcastReceiver:



Public Member Functions

- WiFiDirectBroadcastReceiver (WifiP2pManager manager, Channel channel, WiFiDirectUtils mWiFiDirect↩ Utils)
- void onReceive (Context context, Intent intent)

Private Attributes

- WifiP2pManager manager
- Channel channel
- WiFiDirectUtils mWiFiDirectUtils

6.24.1 Constructor & Destructor Documentation

6.24.1.1 com.copelabs.socio.wifi.WiFiDirectBroadcastReceiver.WiFiDirectBroadcastReceiver (WifiP2pManager *manager,* Channel *channel,* WiFiDirectUtils *mWiFiDirectUtils*)

WiFiDirectBroadcastReceiver constructor.

Parameters

manager	WifiP2pManager system service
channel	WiFi p2p channel
mWiFiDirectUtils	mWiFiDirectUtils class

6.24.2 Member Function Documentation

6.24.2.1 void com.copelabs.socio.wifi.WiFiDirectBroadcastReceiver.onReceive (Context *context,* Intent *intent*)

6.24.3 Member Data Documentation

6.24.3.1 Channel com.copelabs.socio.wifi.WiFiDirectBroadcastReceiver.channel [private]

A channel that connects the application to the Wifi p2p framework.

6.24.3.2 WifiP2pManager com.copelabs.socio.wifi.WiFiDirectBroadcastReceiver.manager [private]

This class provides the API for managing Wi-Fi peer-to-peer connectivity.

6.24.3.3 WiFiDirectUtils com.copelabs.socio.wifi.WiFiDirectBroadcastReceiver.mWiFiDirectUtils [private]

Callback to the WiFiDirectUtils class.

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

- src/com/copelabs/socio/wifi/WiFiDirectBroadcastReceiver.java

6.25 com.copelabs.socio.wifi.WiFiDirectDevice Class Reference

Public Member Functions

- WiFiDirectDevice ()
- void setDevice (WifiP2pDevice device)
- void setInstanceName (String instanceName)
- void setServiceRegistrationType (String serviceRegistrationType)
- void setLastTimeSeen (long mTime)
- WifiP2pDevice getDevice ()
- String getInstanceName ()
- String getServiceRegistrationType ()
- long getLastTimeSeen ()

Private Attributes

- WifiP2pDevice device
- String instanceName = null
- String serviceRegistrationType = null
- long TTL

6.25.1 Detailed Description

A structure to hold service information.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 com.copelabs.socio.wifi.WiFiDirectDevice.WiFiDirectDevice ( )

6.25.3 Member Function Documentation

6.25.3.1 WifiP2pDevice com.copelabs.socio.wifi.WiFiDirectDevice.getDevice ( )

6.25.3.2 String com.copelabs.socio.wifi.WiFiDirectDevice.getInstanceName ( )

6.25.3.3 long com.copelabs.socio.wifi.WiFiDirectDevice.getLastTimeSeen ( )

6.25.3.4 String com.copelabs.socio.wifi.WiFiDirectDevice.getServiceRegistrationType ( )

6.25.3.5 void com.copelabs.socio.wifi.WiFiDirectDevice.setDevice (WifiP2pDevice *device*)

6.25.3.6 void com.copelabs.socio.wifi.WiFiDirectDevice.setInstanceName (String *instanceName*)

6.25.3.7 void com.copelabs.socio.wifi.WiFiDirectDevice.setLastTimeSeen (long *mTime*)

6.25.3.8 void com.copelabs.socio.wifi.WiFiDirectDevice.setServiceRegistrationType (String *serviceRegistrationType*)

6.25.4 Member Data Documentation

6.25.4.1 WifiP2pDevice com.copelabs.socio.wifi.WiFiDirectDevice.device [private]

6.25.4.2 String com.copelabs.socio.wifi.WiFiDirectDevice.instanceName = null [private]

6.25.4.3 String com.copelabs.socio.wifi.WiFiDirectDevice.serviceRegistrationType = null [private]

6.25.4.4 long com.copelabs.socio.wifi.WiFiDirectDevice.TTL [private]

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

- src/com/copelabs/socio/wifi/WiFiDirectDevice.java

6.26 com.copelabs.socio.wifi.WiFiDirectListener Interface Reference

Public Member Functions

- void onPacketReceived (List< Packet > mListOfPackets)
- void onNewWiFiDirectDeviceFound (WiFiDirectDevice mDevice, Map< String, String > mListOfSocial↩ Weight)
- void onWiFiDirectDeviceDisappear (WiFiDirectDevice mDevice)
- void error (int mError)

6.26.1 Member Function Documentation

6.26.1.1 void com.copelabs.socio.wifi.WiFiDirectListener.error (int *mError*)

6.26.1.2 void com.copelabs.socio.wifi.WiFiDirectListener.onNewWiFiDirectDeviceFound (WiFiDirectDevice *mDevice*, Map< String, String > *mListOfSocialWeight*)

6.26.1.3 void com.copelabs.socio.wifi.WiFiDirectListener.onPacketReceived (List< Packet > *mListOfPackets*)

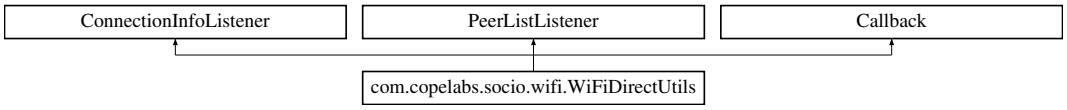
6.26.1.4 void com.copelabs.socio.wifi.WiFiDirectListener.onWiFiDirectDeviceDisappear (WiFiDirectDevice *mDevice*)

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

- src/com/copelabs/socio/wifi/WiFiDirectListener.java

6.27 com.copelabs.socio.wifi.WiFiDirectUtils Class Reference

Inheritance diagram for com.copelabs.socio.wifi.WiFiDirectUtils:



Public Member Functions

- WiFiDirectUtils (Context mContext, WiFiDirect mCallback)
- Handler getHandler ()
- void setHandler (Handler handler)
- void setIsWifiP2pEnabled (boolean isWifiP2pEnabled)
- boolean getIsWifiP2pEnabled ()
- String getWFDirectMacAddress ()
- String getWiFiMacAddress ()
- void start (final String mMAC, final Map< String, String > mListOfSocialWeight)
- void stop ()
- void wifiWasEnabled ()
- void wifiDisabled ()
- void restartDiscovery ()
- void updateRegistration (final String mMAC, final Map< String, String > mListOfSocialWeight)
- void discoverService ()
- void onPeersAvailable (WifiP2pDeviceList peers)
- boolean makeConnection (WiFiDirectDevice mDevice, ArrayList< Packet > mPackets)
- void onConnectionInfoAvailable (WifiP2pInfo p2pInfo)
- void dealWithConnectionEnded (boolean failed)
- void startTimer (String mDeviceMAC)
- void stopTimer ()
- void initializeTimerTask (final String mDeviceMAC)
- boolean handleMessage (Message msg)

Public Attributes

- int WiFiStatus = WIFI_IDLE
- ArrayList< WiFiDirectDevice > mAvailableDevices = new ArrayList<WiFiDirectDevice>()
- String mThisDeviceMACP2p

Static Public Attributes

- static final String SERVICE_INSTANCE = "_oiframework"
- static final String SERVICE_REG_TYPE = "_presence._tcp"
- static final String KEY_MAC = "key_mac"
- static final int WIFI_IDLE = 0
- static final int WIFI_CONNECTING = 1
- static final int WIFI_CONNECTED = 2
- static final int WIFI_DISCONNECTED = 3
- static final int MESSAGE_READ = 0x400 + 1
- static final int SOCKETERROR = 0x400 + 2
- static final int EMPTY = 0x400 + 3

Private Member Functions

- void startRegistrationAndDiscovery (String mMAC, Map< String, String > mListOfSocialWeight)
- void registration (String mMAC, Map< String, String > mListOfSocialWeight)
- void checkNextPacketQueue (String mDeviceMAC)
- void connectP2p (final String mDeviceMAC)
- void disconnectP2p ()
- void notifyPacketReceived (List< Packet > mListOfPackets)
- void notifyOnNewWiFiDirectDeviceFound (WiFiDirectDevice mDevice, Map< String, String > mListOf↩ SocialWeight)
- void notifyOnWiFiDirectDeviceDisappear (WiFiDirectDevice mDevice)
- void notifyError (int mError)

Private Attributes

- Context mContext
- WiFiDirect mCallback
- final IntentFilter intentFilter = new IntentFilter()
- WifiP2pManager manager
- String mSendingToDevice
- Channel channel
- BroadcastReceiver receiver = null
- WifiP2pDnsSdServiceRequest serviceRequest
- WifiP2pDnsSdServiceInfo service
- boolean isWifiP2pEnabled = false
- Map< String, List< Packet > > mToSend = new HashMap<String, List<Packet>>()
- Map< String, String > txtRecordMapReceived
- Handler handler = new Handler(this)
- String gMAC
- Map< String, String > gListOfSocialWeight
- Timer timer
- TimerTask timerTask
- int TimerStatus = TIMER_IDLE

Static Private Attributes

- static final int TIMER_SCHEDULED = 0
- static final int TIMER_IDLE = 1
- static final int TIMER_RUNNING = 2

6.27.1 Constructor & Destructor Documentation

6.27.1.1 com.copelabs.socio.wifi.WiFiDirectUtils.WiFiDirectUtils (Context *mContext,* WiFiDirect *mCallback*)

WiFiDirectUtils constructor.

Parameters

mContext	Interface to global information about an application environment.
mCallback	WiFiDirect module.

6.27.2 Member Function Documentation

6.27.2.1 void com.copelabs.socio.wifi.WiFiDirectUtils.checkNextPacketQueue (String *mDeviceMAC*) [private]

Check if there is more packets inside the queue. If true, starts the timer that will initiate a WiFi P2P connection.

Parameters

mDeviceMAC	Destination device MAC address

6.27.2.2 void com.copelabs.socio.wifi.WiFiDirectUtils.connectP2p (final String *mDeviceMAC*) [private]

Start a WiFi P2P connection.

Parameters

mDeviceMAC	Destination device.

6.27.2.3 void com.copelabs.socio.wifi.WiFiDirectUtils.dealWithConnectionEnded (boolean *failed*)

When a connection finish, this function verifies if an error occurred.

Parameters

failed	error flag.

6.27.2.4 void com.copelabs.socio.wifi.WiFiDirectUtils.disconnectP2p ( ) [private]

Disconnects a WiFi P2P established connection.

6.27.2.5 void com.copelabs.socio.wifi.WiFiDirectUtils.discoverService ( )

Start the WiFi P2P bonjour discover mechanism. A new TXT record is available. Pick up the advertised buddy name.

6.27.2.6 Handler com.copelabs.socio.wifi.WiFiDirectUtils.getHandler ( )

6.27.2.7 boolean com.copelabs.socio.wifi.WiFiDirectUtils.getIsWifiP2pEnabled ( )

Returns the WiFi state.

6.27.2.8 String com.copelabs.socio.wifi.WiFiDirectUtils.getWFDirectMacAddress ( )

Get the MAC address of WiFi P2P interface. Only returns a MAC address if this device is a group owner.

Returns

WiFi P2P interface MAC address

6.27.2.9 String com.copelabs.socio.wifi.WiFiDirectUtils.getWiFiMacAddress ( )

Get the MAC address of WiFi interface.

Returns

WiFi MAC address

6.27.2.10 boolean com.copelabs.socio.wifi.WiFiDirectUtils.handleMessage (Message *msg*)

Handle the output from a WiFi P2P connection.

6.27.2.11 void com.copelabs.socio.wifi.WiFiDirectUtils.initializeTimerTask (final String *mDeviceMAC*)

Timer code that when this timer finish, it start a connection to a specific device.

Parameters

mDeviceMAC	device MAC address to start a WiFi P2P connection.

6.27.2.12 boolean com.copelabs.socio.wifi.WiFiDirectUtils.makeConnection (WiFiDirectDevice *mDevice,* ArrayList< Packet > *mPackets*)

Starts the timer that will initiate a WiFi P2P connection to the destinatio device.

Parameters

mDevice	Destination device
mPackets	List of packets to be sent

Returns

true if connection successfully requested, or false if error occurs.

6.27.2.13 void com.copelabs.socio.wifi.WiFiDirectUtils.notifyError (int *mError*) [private]

Notifies modules above about an error occurred inside the framework

Parameters

mError	Error code.

6.27.2.14 void com.copelabs.socio.wifi.WiFiDirectUtils.notifyOnNewWiFiDirectDeviceFound (WiFiDirectDevice *mDevice,* Map< String, String > *mListOfSocialWeight*) [private]

Notifies the WiFiDirect class that a new device was found.

Parameters

mDevice	Device found
mListOfSocial↩ Weight	List of social weights from the device found

6.27.2.15 void com.copelabs.socio.wifi.WiFiDirectUtils.notifyOnWiFiDirectDeviceDisappear (WiFiDirectDevice *mDevice*) [private]

Notifies the WiFiDirect class that a device disappeared.

Parameters

mDevice	Device found

6.27.2.16 void com.copelabs.socio.wifi.WiFiDirectUtils.notifyPacketReceived (List< Packet > *mListOfPackets*) [private]

Notifies the WiFiDirect class that a new packet arrived.

Parameters

mListOfPackets	Packet arrived

6.27.2.17 void com.copelabs.socio.wifi.WiFiDirectUtils.onConnectionInfoAvailable (WifiP2pInfo *p2pInfo*)

Called when connection info is available.

Parameters

p2pInfo	Connection info

6.27.2.18 void com.copelabs.socio.wifi.WiFiDirectUtils.onPeersAvailable (WifiP2pDeviceList *peers*)

PeerListListener Receives all peers available by WiFi P2P.

Parameters

peers	list of peers available

6.27.2.19 void com.copelabs.socio.wifi.WiFiDirectUtils.registration (String *mMAC,* Map< String, String > *mListOfSocialWeight*) [private]

Registers a new info to be announced by the WiFi P2P

Parameters

mMAC	Local MAC address.
mListOfSocial↩ Weight	List of known SOCIO enabled devices and their social weights.

6.27.2.20 void com.copelabs.socio.wifi.WiFiDirectUtils.restartDiscovery ( )

Restart WiFi P2P Discovery

6.27.2.21 void com.copelabs.socio.wifi.WiFiDirectUtils.setHandler (Handler *handler*)

6.27.2.22 void com.copelabs.socio.wifi.WiFiDirectUtils.setIsWifiP2pEnabled (boolean *isWifiP2pEnabled*)

Set the WiFi status flag

Parameters

isWifiP2pEnabled	the isWifiP2pEnabled to set

6.27.2.23 void com.copelabs.socio.wifi.WiFiDirectUtils.start (final String *mMAC,* final Map< String, String > *mListOfSocialWeight*)

Starts WiFi P2P mechanisms to discover peers and also to register this framework as an available service.

6.27.2.24 void com.copelabs.socio.wifi.WiFiDirectUtils.startRegistrationAndDiscovery (String *mMAC,* Map< String, String > *mListOfSocialWeight*) [private]

Registers a local service and then initiates a service discovery

6.27.2.25 void com.copelabs.socio.wifi.WiFiDirectUtils.startTimer (String *mDeviceMAC*)

Schedule a random timer to avoid collisions during the connection phase.

Parameters

mDeviceMAC	Device MAC address to establish a WiFi P2P connection.

6.27.2.26 void com.copelabs.socio.wifi.WiFiDirectUtils.stop ( )

Stops WiFi P2P mechanism. If a group is formed, it's removed.

6.27.2.27 void com.copelabs.socio.wifi.WiFiDirectUtils.stopTimer ( )

Stops the timer.

6.27.2.28 void com.copelabs.socio.wifi.WiFiDirectUtils.updateRegistration (final String *mMAC,* final Map< String, String > *mListOfSocialWeight*)

Update the info that is announced by the WiFi P2P

Parameters

mMAC	Local MAC address
mListOfSocialWeight	List of known SOCIO enabled devices and their social weight.

6.27.2.29 void com.copelabs.socio.wifi.WiFiDirectUtils.wifiDisabled ( )

When WiFi is disabled, close all local services and service requests.

6.27.2.30 void com.copelabs.socio.wifi.WiFiDirectUtils.wifiWasEnabled ( )

6.27.3 Member Data Documentation

6.27.3.1 Channel com.copelabs.socio.wifi.WiFiDirectUtils.channel [private]

A channel that connects the application to the Wifi p2p framework.

6.27.3.2 final int com.copelabs.socio.wifi.WiFiDirectUtils.EMPTY = 0x400 + 3 [static]

6.27.3.3 Map<String, String> com.copelabs.socio.wifi.WiFiDirectUtils.gListOfSocialWeight [private]

List of known contacts and their social weight announced.

6.27.3.4 String com.copelabs.socio.wifi.WiFiDirectUtils.gMAC [private]

Local MAC address announced

6.27.3.5 Handler com.copelabs.socio.wifi.WiFiDirectUtils.handler = new Handler(this) [private]

Thread handler

6.27.3.6 final IntentFilter com.copelabs.socio.wifi.WiFiDirectUtils.intentFilter = new IntentFilter() [private]

Structured description of Intent values to be matched.

6.27.3.7 boolean com.copelabs.socio.wifi.WiFiDirectUtils.isWifiP2pEnabled = false [private]

WiFi flag about the WiFi state

6.27.3.8 final String com.copelabs.socio.wifi.WiFiDirectUtils.KEY_MAC = "key_mac" [static]

TXT Properties - Key MAC index

6.27.3.9 WifiP2pManager com.copelabs.socio.wifi.WiFiDirectUtils.manager [private]

This class provides the API for managing Wi-Fi peer-to-peer connectivity.

6.27.3.10 ArrayList<WiFiDirectDevice> com.copelabs.socio.wifi.WiFiDirectUtils.mAvailableDevices = new ArrayList<WiFiDirectDevice>()

List of SOCIO enabled devices found by the WiFi P2P.

6.27.3.11 WiFiDirect com.copelabs.socio.wifi.WiFiDirectUtils.mCallback [private]

WiFiDirect module

6.27.3.12 Context com.copelabs.socio.wifi.WiFiDirectUtils.mContext [private]

Interface to global information about an application environment.

6.27.3.13 final int com.copelabs.socio.wifi.WiFiDirectUtils.MESSAGE_READ = 0x400 + 1 [static]

6.27.3.14 String com.copelabs.socio.wifi.WiFiDirectUtils.mSendingToDevice [private]

6.27.3.15 String com.copelabs.socio.wifi.WiFiDirectUtils.mThisDeviceMACP2p

WiFi P2P MAC Address from this device

6.27.3.16 Map<String, List<Packet> > com.copelabs.socio.wifi.WiFiDirectUtils.mToSend = new HashMap<String, List<Packet>>() [private]

Queue of packets to send.

6.27.3.17 BroadcastReceiver com.copelabs.socio.wifi.WiFiDirectUtils.receiver = null [private]

Base class for code that will receive intents sent by sendBroadcast().

6.27.3.18 WifiP2pDnsSdServiceInfo com.copelabs.socio.wifi.WiFiDirectUtils.service [private]

A class for storing Bonjour service information that is advertised over a Wi-Fi peer-to-peer setup.

6.27.3.19 final String com.copelabs.socio.wifi.WiFiDirectUtils.SERVICE_INSTANCE = "_oiframework" [static]

TXT Properties - Application Identification

6.27.3.20 final String com.copelabs.socio.wifi.WiFiDirectUtils.SERVICE_REG_TYPE = "_presence._tcp" [static]

TXT Properties - Application type

6.27.3.21 WifiP2pDnsSdServiceRequest com.copelabs.socio.wifi.WiFiDirectUtils.serviceRequest [private]

A class for creating a Bonjour service discovery request.

6.27.3.22 final int com.copelabs.socio.wifi.WiFiDirectUtils.SOCKETERROR = 0x400 + 2 [static]

6.27.3.23 Timer com.copelabs.socio.wifi.WiFiDirectUtils.timer [private]

Timer used to add a random delay to the WiFi P2P connection to avoid collisions.

6.27.3.24 final int com.copelabs.socio.wifi.WiFiDirectUtils.TIMER_IDLE = 1 [static], [private]

Timer idle status

6.27.3.25 final int com.copelabs.socio.wifi.WiFiDirectUtils.TIMER_RUNNING = 2 [static], [private]

Timer running status

6.27.3.26 final int com.copelabs.socio.wifi.WiFiDirectUtils.TIMER_SCHEDULED = 0 [static], [private]

Timer scheduled status

6.27.3.27 int com.copelabs.socio.wifi.WiFiDirectUtils.TimerStatus = TIMER_IDLE [private]

Timer status

6.27.3.28 TimerTask com.copelabs.socio.wifi.WiFiDirectUtils.timerTask [private]

Timer code.

6.27.3.29 Map<String, String> com.copelabs.socio.wifi.WiFiDirectUtils.txtRecordMapReceived [private]

Map that saves the TXT record received by WiFi P2P.

6.27.3.30 final int com.copelabs.socio.wifi.WiFiDirectUtils.WIFI_CONNECTED = 2 [static]

6.27.3.31 final int com.copelabs.socio.wifi.WiFiDirectUtils.WIFI_CONNECTING = 1 [static]

6.27.3.32 final int com.copelabs.socio.wifi.WiFiDirectUtils.WIFI_DISCONNECTED = 3 [static]

6.27.3.33 final int com.copelabs.socio.wifi.WiFiDirectUtils.WIFI_IDLE = 0 [static]

6.27.3.34 int com.copelabs.socio.wifi.WiFiDirectUtils.WiFiStatus = WIFI_IDLE

WiFi P2P status.

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

- src/com/copelabs/socio/wifi/WiFiDirectUtils.java

6.28 com.copelabs.socio.contentmanager.XmlPullParserHandler Class Reference

Public Member Functions

- List< Packet > getOiMessages ()
- List< Packet > parse (InputStream is)

Private Attributes

- List< Packet > OiMessages = new ArrayList<Packet>()
- Packet OiMessage
- String mTagContent

6.28.1 Detailed Description

Handles the list of messages associated to an xml file

Parameters

OiMessages	list of messages. Each OiMessage holds a set of objects of type Item

Author

rute

Returns

OiMessages

6.28.2 Member Function Documentation

6.28.2.1 List<Packet> com.copelabs.socio.contentmanager.XmlPullParserHandler.getOiMessages ( )

6.28.2.2 List<Packet> com.copelabs.socio.contentmanager.XmlPullParserHandler.parse (InputStream *is*)

Parameters

is	Parses messages stored in an xml file descriptor is

Returns

6.28.3 Member Data Documentation

6.28.3.1 String com.copelabs.socio.contentmanager.XmlPullParserHandler.mTagContent [private]

6.28.3.2 Packet com.copelabs.socio.contentmanager.XmlPullParserHandler.OiMessage [private]

6.28.3.3 List<Packet> com.copelabs.socio.contentmanager.XmlPullParserHandler.OiMessages = new ArrayList<Packet>() [private]

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

- src/com/copelabs/socio/contentmanager/XmlPullParserHandler.java

Chapter 7

File Documentation

7.1 src/com/copelabs/socio/bt/BluetoothManager.java File Reference

Classes

- class com.copelabs.socio.bt.BluetoothManager
- class **com.copelabs.socio.bt.BluetoothManager.FindBluetoothDevice**
- class **com.copelabs.socio.bt.BluetoothManager.adapterBroadcastReceiver**

Packages

- package com.copelabs.socio.bt

7.2 src/com/copelabs/socio/bt/BTDeviceFinder.java File Reference

Classes

- interface com.copelabs.socio.bt.BTDeviceFinder

Packages

- package com.copelabs.socio.bt

7.3 src/com/copelabs/socio/contentmanager/CleanTask.java File Reference

Classes

- class com.copelabs.socio.contentmanager.CleanTask

Packages

- package com.copelabs.socio.contentmanager

7.4 src/com/copelabs/socio/contentmanager/ContentManager.java File Reference

Classes

- class com.copelabs.socio.contentmanager.ContentManager

Packages

- package com.copelabs.socio.contentmanager

7.5 src/com/copelabs/socio/contentmanager/FileIO.java File Reference

Classes

- class com.copelabs.socio.contentmanager.FileIO

Packages

- package com.copelabs.socio.contentmanager

7.6 src/com/copelabs/socio/contentmanager/Packet.java File Reference

Classes

- class com.copelabs.socio.contentmanager.Packet

Packages

- package com.copelabs.socio.contentmanager

7.7 src/com/copelabs/socio/contentmanager/XmlPullParserHandler.java File Reference

this class corresponds to the core of Oi!, an instant messaging tool for delay tolerant networking Oi can work without an infrastructure Oi performs routing of messages based on the dLife opportunistic solution

Classes

- class com.copelabs.socio.contentmanager.XmlPullParserHandler

Packages

- package com.copelabs.socio.contentmanager

7.7.1 Detailed Description

this class corresponds to the core of Oi!, an instant messaging tool for delay tolerant networking Oi can work without an infrastructure Oi performs routing of messages based on the dLife opportunistic solution

COPELABS - Oi!

Author

Rute Sofia (COPELABS/ULHT)

Date

20/05/2015

Version

v1.0 - pre-prototype

7.8 src/com/copelabs/socio/router/Routing.java File Reference

Classes

- class com.copelabs.socio.router.Routing

Packages

- package com.copelabs.socio.router

7.9 src/com/copelabs/socio/router/RoutingListener.java File Reference

Classes

- interface com.copelabs.socio.router.RoutingListener

Packages

- package com.copelabs.socio.router

7.10 src/com/copelabs/socio/socialproximity/DataBase.java File Reference

Classes

- class com.copelabs.socio.socialproximity.DataBase

Packages

- package com.copelabs.socio.socialproximity

7.11 src/com/copelabs/socio/socialproximity/DataBaseChangeListener.java File Reference

Classes

- interface com.copelabs.socio.socialproximity.DataBaseChangeListener

Packages

- package com.copelabs.socio.socialproximity

7.12 src/com/copelabs/socio/socialproximity/OnSocialWeightUpdate.java File Reference

Classes

- class com.copelabs.socio.socialproximity.OnSocialWeightUpdate

Packages

- package com.copelabs.socio.socialproximity

7.13 src/com/copelabs/socio/socialproximity/SocialProximity.java File Reference

Classes

- class com.copelabs.socio.socialproximity.SocialProximity
- class com.copelabs.socio.socialproximity.SocialProximity.CustomComparator
- class **com.copelabs.socio.socialproximity.SocialProximity.ServiceBTListener**

Packages

- package com.copelabs.socio.socialproximity

7.14 src/com/copelabs/socio/socialproximity/SocialProximityListener.java File Reference

Classes

- interface com.copelabs.socio.socialproximity.SocialProximityListener

Packages

- package com.copelabs.socio.socialproximity

7.15 src/com/copelabs/socio/socialproximity/SocialWeight.java File Reference

Classes

- class com.copelabs.socio.socialproximity.SocialWeight

Packages

- package com.copelabs.socio.socialproximity

7.16 src/com/copelabs/socio/socialproximity/SQLiteHelper.java File Reference

Classes

- class com.copelabs.socio.socialproximity.SQLiteHelper

Packages

- package com.copelabs.socio.socialproximity

7.17 src/com/copelabs/socio/socialproximity/UserDevAverageEncounterDuration.java File Reference

Classes

- class com.copelabs.socio.socialproximity.UserDevAverageEncounterDuration

Packages

- package com.copelabs.socio.socialproximity

7.18 src/com/copelabs/socio/socialproximity/UserDevEncounterDuration.java File Reference

Classes

- class com.copelabs.socio.socialproximity.UserDevEncounterDuration

Packages

- package com.copelabs.socio.socialproximity

7.19 src/com/copelabs/socio/socialproximity/UserDeviceInfo.java File Reference

Classes

- class com.copelabs.socio.socialproximity.UserDeviceInfo

Packages

- package com.copelabs.socio.socialproximity

7.20 src/com/copelabs/socio/socialproximity/UserDevSocialWeight.java File Reference

Classes

- class com.copelabs.socio.socialproximity.UserDevSocialWeight

Packages

- package com.copelabs.socio.socialproximity

7.21 src/com/copelabs/socio/wifi/ClientSocketHandler.java File Reference

Classes

- class com.copelabs.socio.wifi.ClientSocketHandler

Packages

- package com.copelabs.socio.wifi

7.22 src/com/copelabs/socio/wifi/ServerThread.java File Reference

Classes

- class com.copelabs.socio.wifi.ServerThread

Packages

- package com.copelabs.socio.wifi

7.23 src/com/copelabs/socio/wifi/WiFiDirect.java File Reference

Classes

- class com.copelabs.socio.wifi.WiFiDirect

Packages

- package com.copelabs.socio.wifi

7.24 src/com/copelabs/socio/wifi/WiFiDirectBroadcastReceiver.java File Reference

Classes

- class com.copelabs.socio.wifi.WiFiDirectBroadcastReceiver

Packages

- package com.copelabs.socio.wifi

7.25 src/com/copelabs/socio/wifi/WiFiDirectDevice.java File Reference

Classes

- class com.copelabs.socio.wifi.WiFiDirectDevice

Packages

- package com.copelabs.socio.wifi

7.26 src/com/copelabs/socio/wifi/WiFiDirectListener.java File Reference

Classes

- interface com.copelabs.socio.wifi.WiFiDirectListener

Packages

- package com.copelabs.socio.wifi

7.27 src/com/copelabs/socio/wifi/WiFiDirectUtils.java File Reference

Classes

- class com.copelabs.socio.wifi.WiFiDirectUtils

Packages

- package com.copelabs.socio.wifi