Action full title:

Universal, mobile-centric and opportunistic communications architecture

Action acronym:

UMOBILE

Deliverable:

D3.2. “UMOBILE architecture report (2)”

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, Athena Research Center

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	D3.2 UMOBILE architecture report (2)
WP Number	WP3
WP Leader	Sotiris Diamantopoulos
Task Leader	Sotiris Diamantopoulos
Authors	**ATHENA**: Sotiris Diamantopoulos, Christos-Alexandros Sarros, Dimitris Vardalis, Konstantinos Prassopoulos, Vassilis Tsaoussidis **COPELABS**: Paulo Mendes, Seweryn Dynerowicz, Omar Aponte **UCL:** Ioannis Psaras, Sergi Rene **UCAM:** Adisorn Lertsinsrubtavee, Carlos Molina-Jimenez, Arjuna Sathiaseelan **TECNALIA:** Susana Sanchez Perez **SENCEPTION**: Rute Sofia
Contact	diamantopoulos.sotiris (at) gmail.com , csarros (at) ee.duth.gr
Due date	31/7/2017
Actual date of submission	3/8/2017

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	

Document History:

Version	Date	Description
1.0	14/07/17	First draft of the final architecture
1.1	3/08/17	Submission to the EC

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 within the framework of the UMOBILE project WP3 "System and node architecture development". The deliverable aims to describe the final UMOBILE architecture, accompanied by the implementation of the architecture 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 ICN, DTN and opportunistic networking, we establish an architectural framework that extends connectivity options by being delay-tolerant and by 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 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. The result is a novel architecture that allows for new services and applications.

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 as part of the UMOBILE architecture. The code that we have implemented is also discussed and made available from the UMOBILE project public GitHub repository.

In the rest of the document, we present our vision and provide the full architectural picture, as well as the specific modules and their interconnections.

NOTE (to be removed from the public version):
D3.2 provides the detailed view of the UMOBILE architecture, while D3.4 is a high-level description. Therefore, it is suggested that the reader/reviewer starts from D3.4 first, in order to get the high-level overview of the architecture and then go into the details in D3.2.

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

1. Table of 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

List of Definitions

Term	Meaning
AP	An access point (AP) is a networking hardware device that allows a WiFi compliant device to connect to a wired network.
Application	Computer software design to perform a single or several specific tasks, e.g. a calendar and map services.
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.
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.

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

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-prefix based routing protocol, and can have multiple output interfaces for each prefix.
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.
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 efficient 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 flexibility 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.

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

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.
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 *Daemon*
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 a communication and interaction sensing platform that performs, among others, seamless roaming contextualization of a user daily routine.
PIT	A table that stores all the Interests that a router has forwarded but not satisfied yet. Each PIT entry records the data name carried in the Interest, its incoming and outgoing interface.
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

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

	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 service migration module.
Service	Service refers to a computational operation or application running on the network which can fulfil an end user's request. The services can be hosted and computed in some specific nodes such as servers or gateways. Specifically, 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 definition; "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; affinities between end 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 and services to end-users. By efficiently we mean that content/services are reliably available, with the expected quality of service and despite any impairments of the communication infrastructure. 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 delays and 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 while they are trying to access Internet services.

To achieve ubiquitous, local and edge-based services, 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 the UMOBILE architecture in detail. We place special focus on the new modules developed in order to:

- Achieve the integration of ICN and DTN, to enable delay-tolerant and opportunistic communications in an information-centric framework.
- Allow for new services at the edge of the network.
- Assist in usage contextualisation to develop new types of applications.
- Consolidate the architecture with novel forwarding, and routing mechanisms.
- Provide the architecture with three QoS mechanisms that operate either individually or collaboratively at different levels of the software stack: service migration platform that operates and the application level and DTN-framework and Flowlet congestion control (INRPP) that operate at the network layer.

We analyse the individual components in the following sections. Our goal is to integrate all modules into a unified UMOBILE platform that will provide the aforementioned functionality.

As detailed at D3.4, the high-level description of the UMOBILE architecture, the proposed UMOBILE architecture consists of two main parts: elements in the fixed part of the network (e.g., routers, gateways, access points) and mobile devices (focusing mainly on Android).

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

Apart from describing all UMOBILE architectural components, the present deliverable also includes all the code developed within the framework of the UMOBILE project, available in the following link:

https://github.com/umobileproject/

The document is organised as follows:

- Section 2 provides an insight to the vision of UMOBILE and highlights how each UMOBILE feature contributes to this vision,
- Section 3 provides an introduction to Named Data Networking, as the starting point of the UMOBILE architecture.
- Section 4 presents the full architectural picture, along with the new services and mechanisms developed as part of the UMOBILE platform
- Section 5 provides the native applications developed to run over the UMOBILE platform
- Section 6 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 vision

In D3.4, we describe the UMOBILE architecture and highlight the key services that the proposed architecture will provide. Figure 1 (extracted from D5.1) provides an overview of the UMOBILE platform, in terms of actors and connectivity options involved.

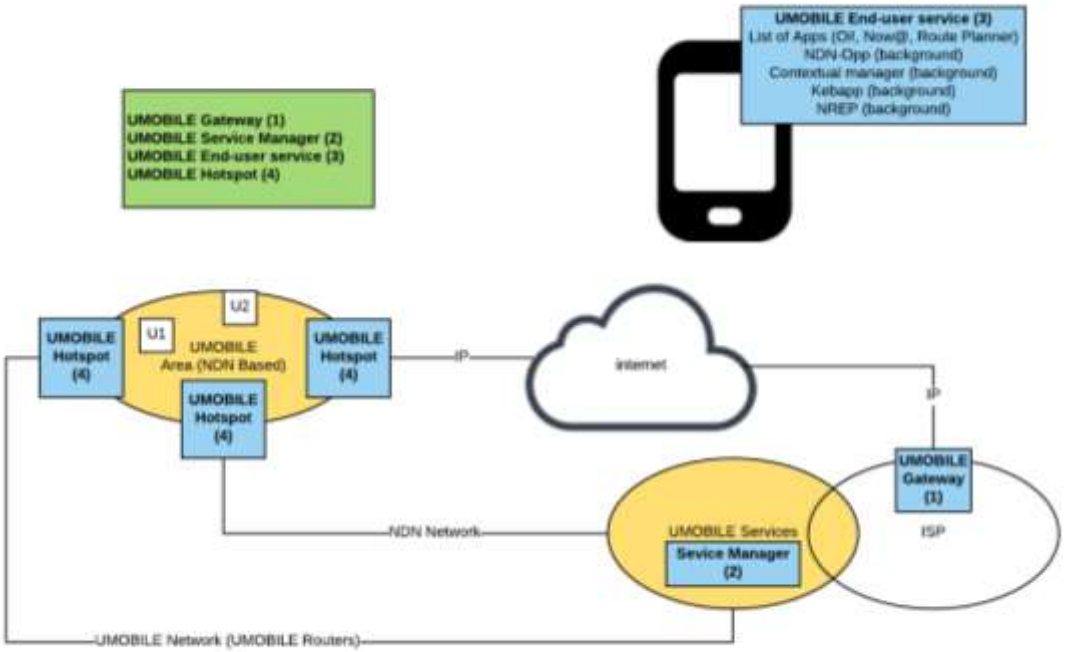

Figure 1: High-level design of the UMOBILE architecture

The actors involved in the UMOBILE architecture include:

- **UMOBILE Hotspots**: collect and relay relevant information, host some instantiated services or store collected data, check its validity and perform computational functions to increase the value of the information to the civil authorities. Some of these hotspots will be isolated access points (APs) providing specific services
- **UMOBILE gateway**: provide interconnectivity between UMOBILE domain and the Internet domain.
- **UMOBILE service manager**: implement the functionality that the Service Provider needs to deploy his services.
- **UMOBILE end-user service:** send and receive, as well as carry and forward data, based on an opportunistic networking approach.

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

The aim of the UMOBILE project is to provide an architecture that merges information-centric networking with delay-tolerant networking to efficiently operate in different network situations, reaching disconnected environments and users and providing new types of services. These services can be exploited from a variety of applications (examples of which include opportunistic chat and news apps). As the envisioned UMOBILE services are detailed in D3.4, 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:

- Build a new delay-tolerant face in the NDN architecture. This DTN face is used in disruptive and opportunistic scenarios and abstracts the communication specifics from NDN by delivering packets to an underlying Bundle Protocol implementation (IBR-DTN[1]). The latter is responsible for their delay-/disruption-tolerant transmission;
- Develop a novel framework (NDN-Opp) that natively enables opportunistic communications over NDN and supports social-aware routing. In this context, a new WiFi Direct face is introduced;
- Provide two new types of services that can be used in a variety of scenarios: i) a service migration platform that can be used by service providers to deploy services of different QoS classes such as premium, best-effort and less-than-best effort. Services are normally deployed closer to the edge of the network where they are more likely to satisfy the expected QoS commitments. For example, a service deployed at the edge is likely to guarantee fast response time, even when remote server located at the core of the network are not reachable; and ii) a push service that can be used in situations where the user needs to push data to the network (e.g. in areas struck by disaster events, such as fires and floods);
- Propose a new application-centric naming framework (KEBAPP) [9], where applications share common name-spaces and further support the use of a keywords;
- Employ novel mechanisms to cope with extreme situations (e.g. disaster situations): an opportunistic off-path content discovery mechanism that exploits cached content in both in-network caches and end-users' devices to facilitate content retrieval even when the original server is unreachable, as well as a name-based replication priorities scheme (NREP)[4] that favours spreading of the most important messages when normal communications are disrupted and traffic is increased.
- Introduce a new congestion control algorithm (INRPP) [15] which pools bandwidth and in-network cache resources and can prioritise traffic in a novel congestion control framework to reach global fairness and local stability. Integrate the contextual manager to assist in contextualization and thus rely on data mining to improve aspects of the network operation, such as, but not limited to, routing, or service management.

[1] https://www.ibr.cs.tu-bs.de/projects/ibr-dtn/

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

- Propose an NDN routing approach based on contextualization, to assist in extending current NDN routing into opportunistic environments.

An overview of the different components and their contributions to the architecture is given in the image below:

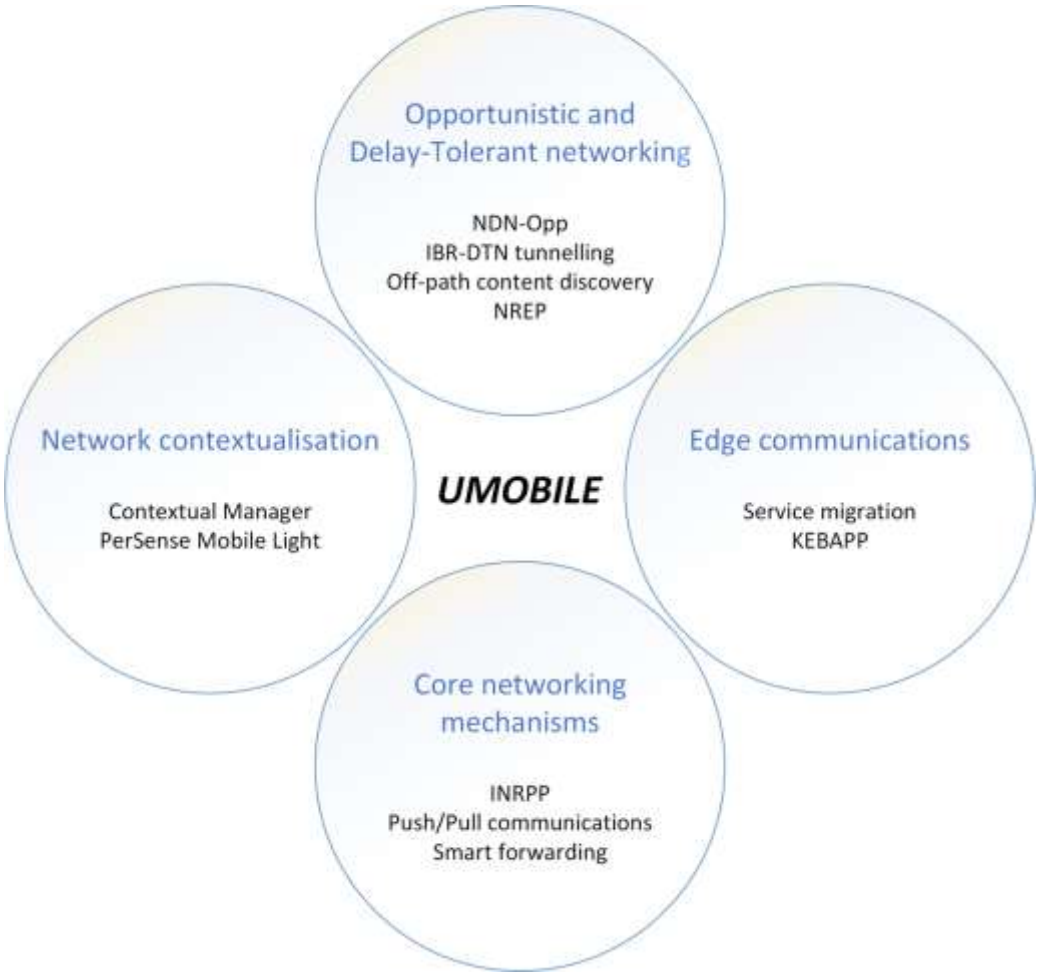

Figure 2: UMOBILE architecture conceptual overview

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

3. UMOBILE starting points

The aim of UMOBILE is to build an architecture that defines a new service abstraction, bringing together information-centric, delay-tolerant, and opportunistic communications principles into one single abstraction. To achieve that, UMOBILE uses Named Data Networking (NDN) - one of the most promising ICN implementations - as a starting point. We modify and enhance the NDN architecture in order to naturally support delay-tolerant and opportunistic communications, building an integrated platform that facilitates communications in disruptive environments at the edge of the network.

In this Section, we provide a brief description of NDN, which constitutes the basis for the UMOBILE architecture.

Named Data Networking (NDN)

Named Data Networking (NDN)[2] is a Future Internet architecture that aims to transition today's host-centric network architecture into a data-centric network architecture. Users will no longer need to retrieve data from a specific physical location; instead, they 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 identified by a given name; NDN is a part of the broader Information-Centric Networking (ICN) approaches where 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

[2] https://named-data.net

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

provide a solid base to build an architecture for mobile, infrastructureless or delay tolerant networks.

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.

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. In order to provide connectivity between any opportunistic UMOBILE device and the fixed network (IP network), we also 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. There is an one-to-one correspondence between an Interest and a Data packet; both types of packets carry a name that identifies a piece of data, while the Interest packet can be interpreted as a request for the specific Data packet by the consumer. More specifically:

- **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 the 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 yet satisfied. Each PIT entry records the data name carried in the Interest, its incoming and outgoing interface.
- **Forwarding Information Base (FIB)**: A forwarding table which maps name components to interfaces. The FIB itself is populated by a name-prefix based routing protocol, and can have multiple output interfaces for each prefix.

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

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

Figure 3 depicts the high-level design of NDN architecture.

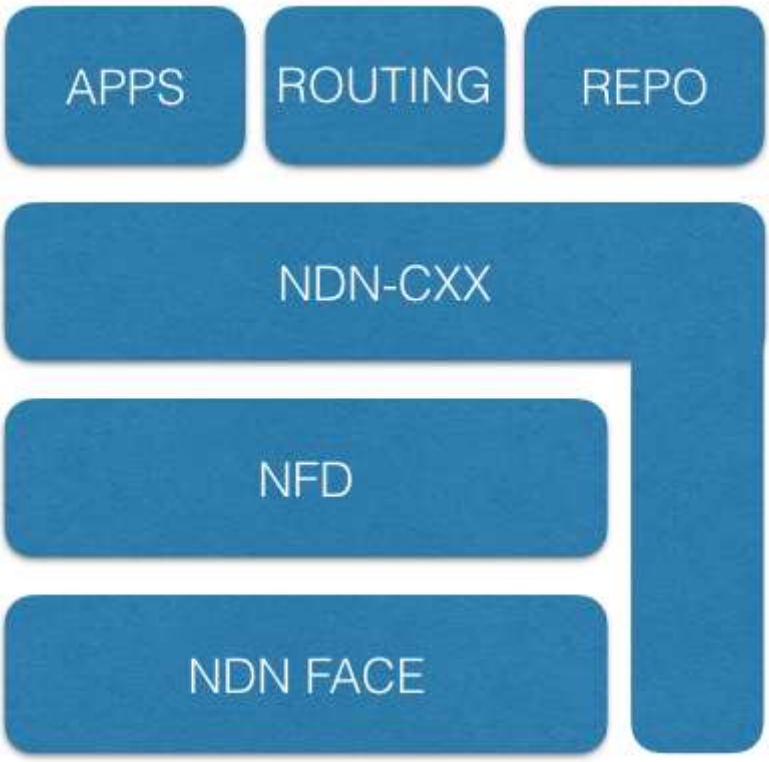

Figure 3: Conceptual depiction of Named Data Networking architecture

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

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

4. UMOBILE architecture

The UMOBILE architecture builds on the NDN platform. We have modified and expanded the original NDN platform to enable new services while focusing on the edge of the network, especially on opportunistic mobile communications. Our goal is to exploit all communication opportunities, achieving reliability and resilience even against extremely challenging network conditions where connectivity to the core of the network is not to be taken for granted.

QoE for the end-user is improved by utilising service-centric networking principles to deploy services at the edge of the network, as well as via user, usage and network contextualisation. In addition to the aforementioned benefits, contextualization assists in the development of new strategies, e.g. "smart" forwarding strategies which shall consider not only the network context, as well as the user networking context to improve the network operation.

In addition, we exploit the properties of Information-Centric and Delay-Tolerant Networks to design novel congestion control mechanisms, QoS schemes and make relevant advancements to improve core network operation.

Our starting point is – as already noted – the original NDN architecture. Below, we provide an abstraction of the modules present on the original NDN platform along with their interrelations.

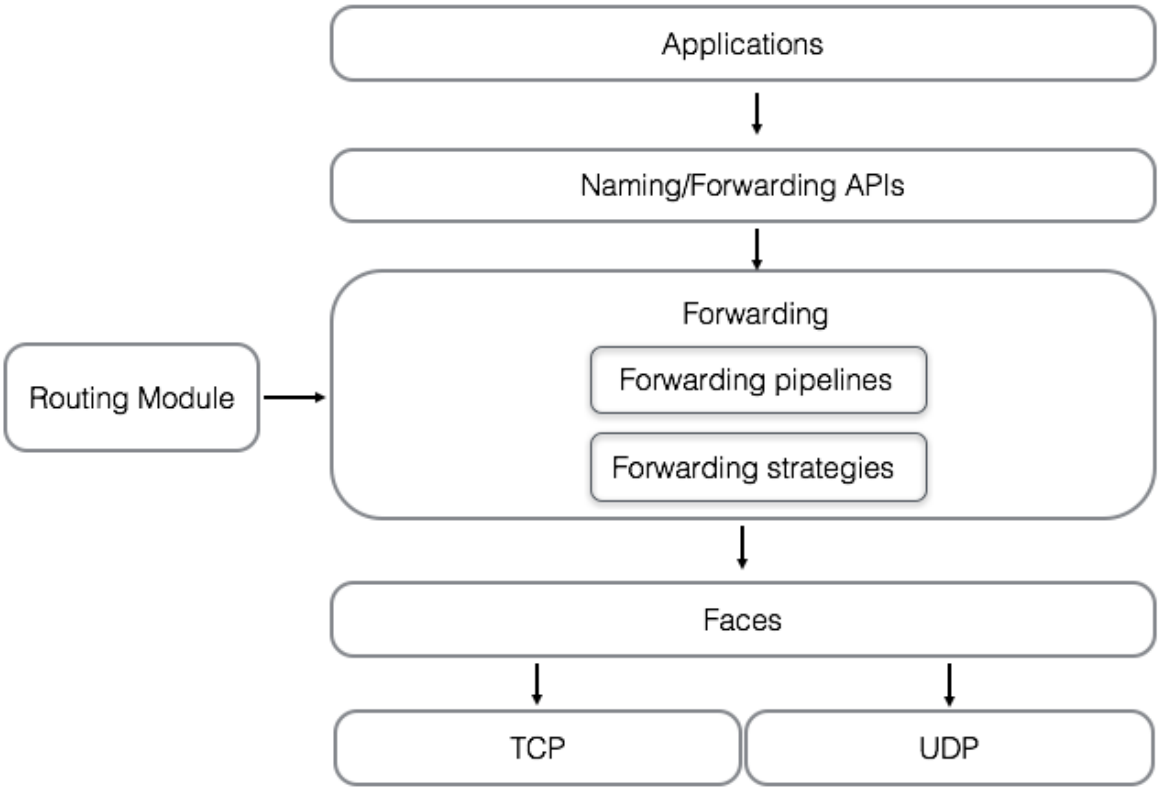

Figure 4: NDN architectural modules

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

Software-wise, the UMOBILE platform extends NDN providing support for decentralised services and opportunistic communications. Furthermore, newly-developed applications, running natively over the UMOBILE platform, take advantage of the push-communication model supported by the UMOBILE platform, to provide end-users with novel services.

The full architectural diagram of the UMOBILE platform is provided in Fig. 5. Orange boxes depict new components implemented in the framework of the UMOBILE project, whereas blue ones depict modified components of the original NDN platform

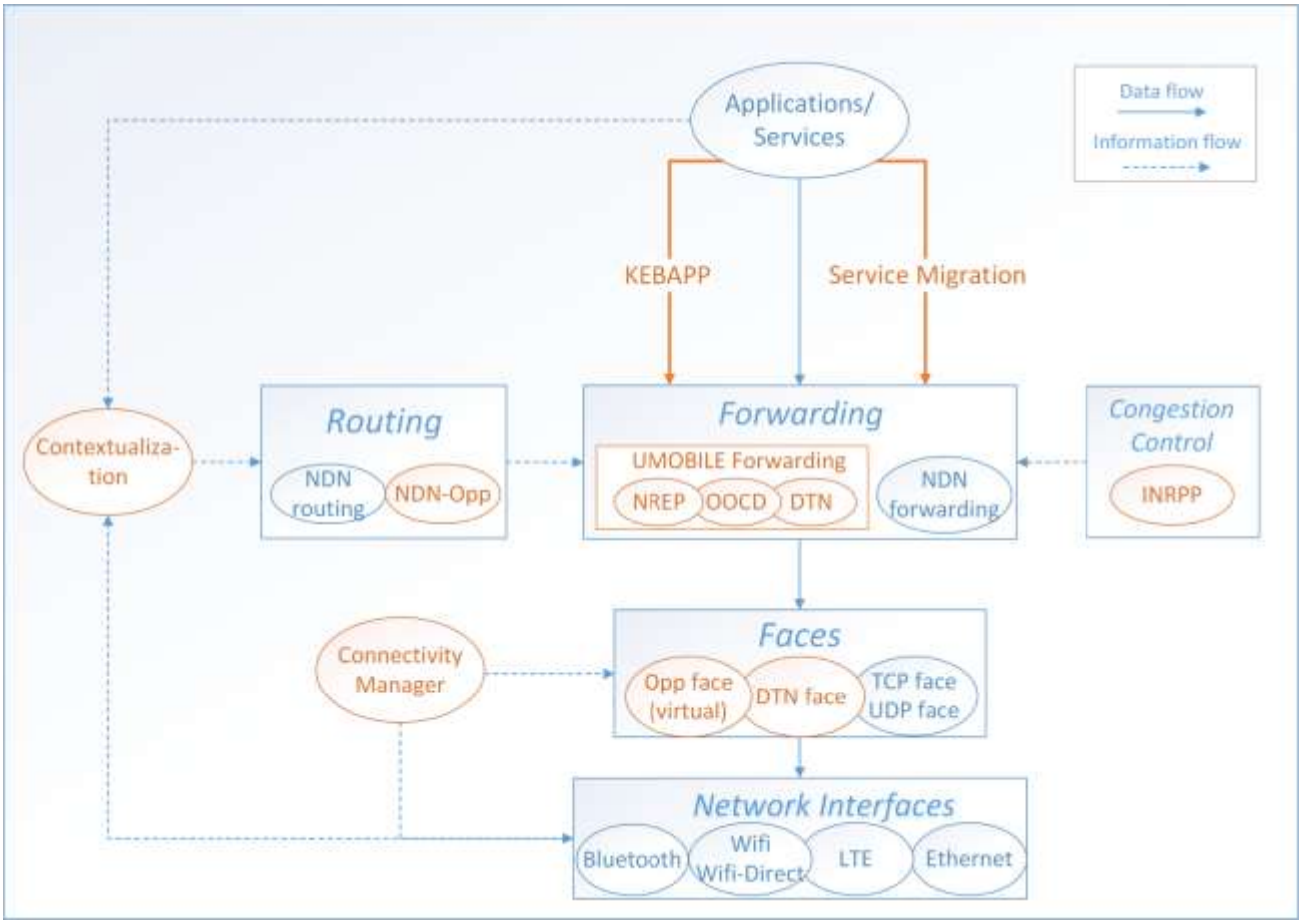

Figure 5 : UMOBILE architectural modules

New and updated software modules include:

a) New forwarding mechanisms

- DTN forwarding engine

  We have implemented and integrated a DTN interface into the UMOBILE platform to enable the forwarding of packets through DTN tunnelling. DTN forwarding provides enhanced resilience in less than ideal networking conditions. It can also be employed as a QoS class.

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

- Name-based Replication Priorities (NREP)

  NREP prioritises replication of packets based on their naming characteristics.

- Off-Path Content Discovery (OOCD)

  Off-Path Content Discovery extends typical NDN forwarding providing alternative content sources.

These forwarding strategies are developed in the framework of T3.3, except NREP, which is developed in the framework of T4.3.

b) An application sharing framework (KEBAPP)

KEBAPP is an application sharing framework that enables the discovery of locally available application data, without the need to contact a centralized service.

c) Support for opportunistic networks

Support for opportunistic networks is being developed within the UMOBILE project specifically to enable the use of NDN. In such types of networks, no infrastructure is required for devices to exchange data among each other as they can use device-to-device communications to perform these exchanges directly. Section 4.6 describes our implementation for this type of scenarios; NDN-Opp. Furthermore, all content generated among the devices can be accessed by passing through others in a multi-hop manner and not rely on access to the Internet.

d) Support for push communications

UMOBILE extends the NDN communication model by supporting push communications. This support comes in the form of two extensions each suited for one type of scenarios. The first extension enables applications to send several pieces of content to a certain device during a certain period of time. This extension is more suited for usual messaging types of applications. The second extension allows a device to produce and forward content to a device without the need for any Interest to be sent in the first place. This extension is more suited for scenarios where it is inconceivable to rely on an Interest mechanism for sending messages, for example for emergency applications.

e) A multi-plane QoS mechanism

A multi-planar QoS mechanism is being developed in the framework of the UMOBILE project. We have implemented a service migration platform that operates at the application level and can be used by a service provider for deploying services that are expected to observe QoS requirements. The service migration platform can be used as a reactive mechanism and as proactive mechanism. As a reactive mechanism the service migration platform operates at service delivery time. It monitors the current status of the resources (e.g., hotspots and containers used for running the services) used for running the services and reacts by deploying additional instances of a service when the current ones show signs of exhaustion. As a proactive mechanism the

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

service migration platform operates independently of the actual instantiation of the services. It anticipates service requests, transfers images of the service at convenient time (e.g., midnight) to caches where they are likely to be needed in the near future and leaves them ready for immediate instantiation.

Complimentary, at the network level, INRPP takes advantage of in-network caching and multiple path availability to mitigate congestion to observe QoS requirements. Finally, DTN forwarding is employed either to overcome network impairments, or to provide a lower-than-best-effort class, leveraging alternative opportunistic links.

f) A new Contextualization Agent

The contextual manager is a UMOBILE module that resides on end-user devices and that captures both external (roaming) and internal (usage) data, to assist a better network operation (e.g. opportunistic routing; priorities in terms of resource management; interest match in terms of data dissemination). The contextual manager is a novel tool that relies on seamless data capture and data mining to assist in improving the network operation.

g) New applications

- Oi!. an application which allows the users to exchange messages independently of the availability of Internet access, by exploiting the direct wireless communications capabilities (i.e., Bluetooth and Wi-Fi direct) available in personal mobile devices.
- Now@ ,an open-source application developed for Android that allows users to exchange information such as text, images and documents over an NDN infrastructure.
- PerSense Mobile Light (PML). PML is an example of an external application that the Contextual Manager can be plugged to, to collect data. The purpose is to show how the Contextual Manager can be easily extended to collect new parameters, by relying on external applications.

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

4.1. Forwarding

Forwarding in UMOBILE follows typical NDN forwarding, by enhancing it with new forwarding strategies and modifications that allow for delay/disruption tolerant and opportunistic communications. We aim to provide support for challenging network scenarios with intermittent connectivity, by enabling D2D communications over WiFi Direct and Bluetooth links, or even remote WiFi hotspots.

We note that forwarding plays a special role in the UMOBILE architecture. In traditional IP routing, routers exchange updates and select the best routes to construct the forwarding table (FIB) but the forwarding plane by itself strictly follows the FIB. This results in a "smart routing/dumb forwarding" approach that places the responsibility for data delivery on the routing.

In UMOBILE, and following the NDN architecture, this relationship is changed. While routing serves the same purpose (to compute routing tables to be used in forwarding NDN's Interest packets), the forwarding plane is now stateful: the routers keep state of the pending Interests to guide Data packets back to requesting consumers. This allows for an intelligent forwarding plane, as by recording pending Interests and observing Data packets coming back, each NDN router can measure packet delivery performance and utilise multiple alternative paths to improve performance. As a result, much of the routing functionality is offloaded to forwarding Interest packets and the routing plane only needs to disseminate long-term changes in topology, without having to deal with short-term churns.

Forwarding strategies are the decision makers in the forwarding *daemon*, deciding whether, when and where to forward the Interests. A per-namespace selection of the appropriate strategy is available, to fulfil the needs for different applications that utilise different naming schemes. By utilising different naming schemes, we can invoke different strategies and provide support for multiple applications.

Below we detail the modules included in the UMOBILE architecture that are related to the forwarding plane. Our major focus is on extending the reach of the communication paradigm, by enabling and facilitating delay-tolerant and opportunistic communications that are currently poorly supported by the NDN paradigm.

4.1.1. DTN tunnelling

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:

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

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.

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.

A new face connecting the UMOBILE platform with the IBR-DTN *daemon* has been developed in the framework of the project. This allows the *natural* integration of NDN and DTN networking platforms: it enables a whole set of new communication capabilities when necessary. In particular, DTN enhances network operation by providing:

i) A new service class, namely less-than-best-effort. This shall support services with non-stringent QoS requirements.
ii) Reliability to services in congested environments (e.g. in emergency cases)
iii) A means of congestion control, by reactively offloading traffic to DTN when congestion is detected.
iv) A means of congestion avoidance, by proactively scheduling and/or shaping traffic to reach its destination only when links are underutilized.

Architecturally, the DTN face is the module interconnecting the UMOBILE platform with the IBR-DTN *daemon*, as depicted in Fig 6.

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

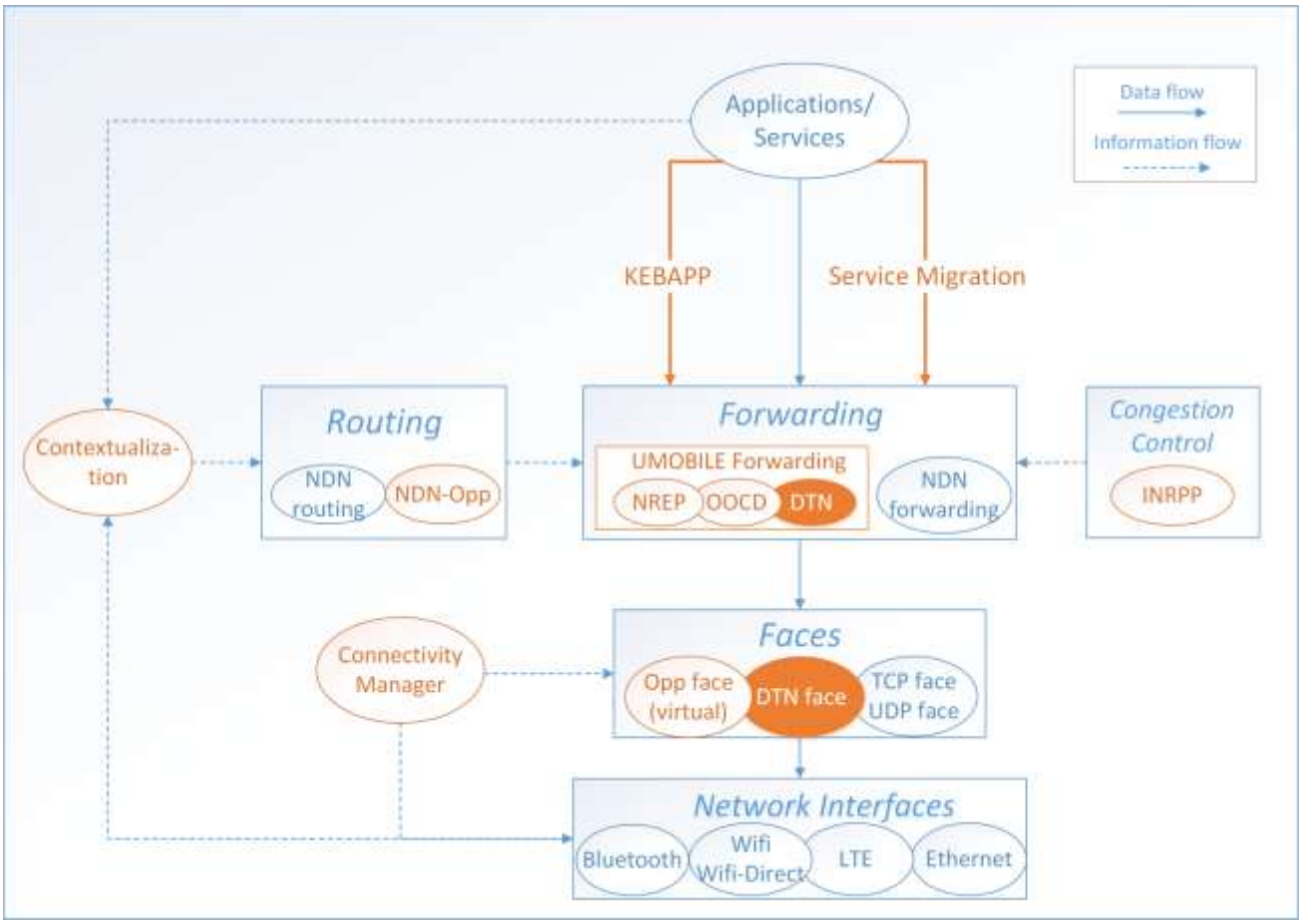

Figure 6. NDN-DTN tunnelling

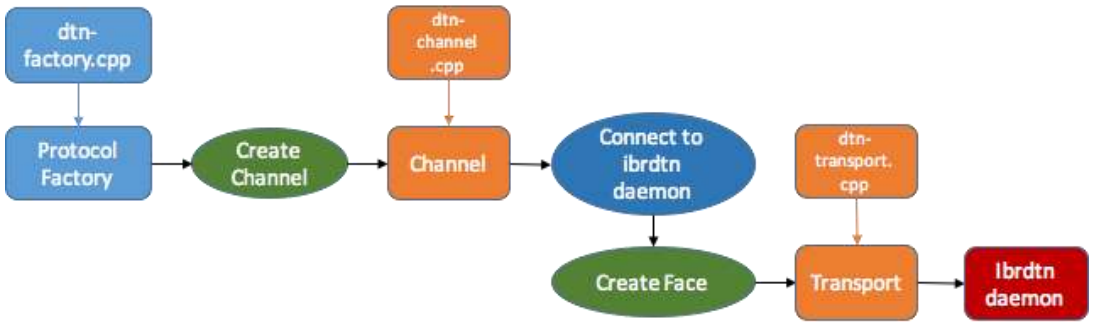

Figure 7. Internal structure of the Linux DTN face

The DTN face is employed in the forwarding plane as follows:

- In the non-infrastructure part of the network, it can be employed by forwarding strategies to opportunistically reach another, possibly more "fixed" node (e.g. an access point in the vicinity), instead of relying on typical NDN over TCP/UDP forwarding. This functionality is depicted in Fig. 8

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

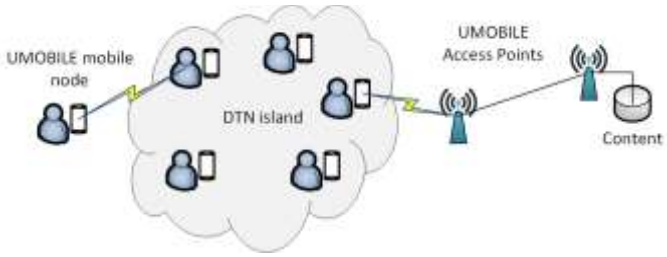

Figure 8. DTN forwarding in volatile networking environments

- It can be employed a priori by applications when network delays are to be expected (e.g. there is content delivery through UAVs), in an opportunistic or infrastructure network context. This can be achieved using the client control forwarding strategy, having the consumer application specify DTN as the appropriate next-hop face for the intermediate nodes. This functionality is depicted in Fig. 9

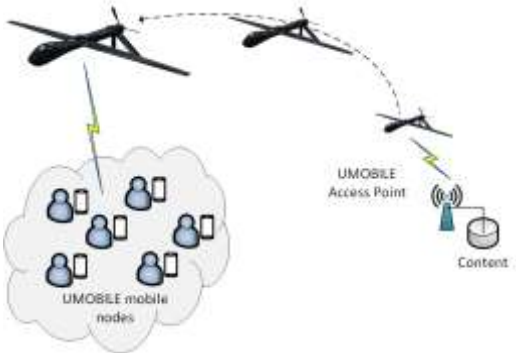

Figure 9. DTN forwarding accommodating delays inflicted by a UAV flying pattern

- It will be a part of a QoS scheme supporting a) services with non-stringent QoS requirements (lower-than-best-effort QoS class), as depicted in Fig. 10, or b) services that prioritise reliability in congested environments (e.g. in emergency cases), depicted in Fig. 11.

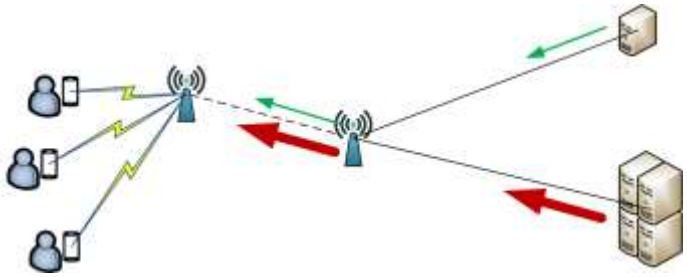

Figure 10. DTN forwarding providing a lower QoS class

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

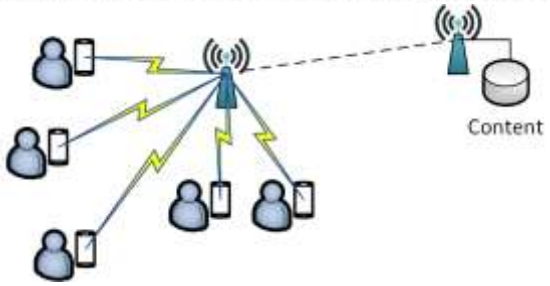

Figure 11. DTN forwarding increasing reliability in congested environments

The DTN forwarding engine is employed either proactively by the application (e.g. by the use of the client-control strategy), or reactively, upon reception of congestion-related messages.

In terms of system requirements, the DTN integration into the UMOBILE platform covers the following requirements:

- R2: UMOBILE systems MUST be able to exchange data by exploiting every communication opportunity through WiFi (structured, direct), 3G and Bluetooth, among UMOBILE systems, operating even in situations with intermittent Internet connectivity)
- R13: UMOBILE systems MUST be able to provide the services to the end users when there is no Internet connectivity)

Code Documentation

Linux

NDN consists of:

1. *ndn-cxx* library, which provides infrastructure facilities for the system,
2. *ndn*, which includes the actual forwarding daemon (NFD) and several related tools

The following contributions have been made to realise the Linux NDN-DTN integration, depicted in Fig.7:

- Library *ndn-cxx_umobile* contributions:
    1. Extended the FaceURI class so that it can handle DTN faces.
        a. Added a "dtn" scheme that takes as the endpointPrefix as the host (e.g. dtn-node1) and the endpointAffix as the path (e.g. /nfd)
        b. Added a DTNCanonizeProvider that checks if a certain dtn *URI* is canonical. Currently all *URI*s are considered canonical.

- Daemon ndn-dtn contributions:

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

1. Added section in the *FaceManager* that initializes the DTN Face from the *nfd.conf* configuration file. The configuration allows for setting the host and port for the ibrdtn daemon and the endpoint prefix and affix for ibrdtn.
    a. Normally the ibrdtn daemon host is the localhost, but a remote host could also be used as a gateway
    b. The *endpoint id* of the local dtn host is in the form of dtn://dtn-node1/nfd, where "dtn" is the scheme, "dtn-node1" is the endpointPrefix and nfd is the endpointAffix. nfd is the dtn application endpoint that the nfd daemon will subscribe to in order to receive ndn-related bundles.
2. Created DtnFactory class, responsible for creating and maintaining the DtnChannel. When the daemon initializes, a DtnChannel is created. Subsequent invocations of the createChannel function of the DtnChannel return the already created channel.
3. Created an AsyncIbrDtnClient for asynchronously connecting to the ibrdtn deamon. The client starts a new thread that runs in the background and is always connected to the dtn deamon. When a new bundle arrives, the client notifies the associated DtnChannel.
4. Created DtnChannel that receives and sends data. DtnChannel listens for incoming data and forwards them to the daemon through the relevant face and also sends outgoing data, again through the relevant face.
    a. Each DtnChannel creates an AsyncIbrDtnClient in the "listen" function and passes itself as a constructor argument. When a new bundle arrives the client calls the relevant callback at the DtnChannel. The client thread remains on for the entire lifetime of the channel and gets deleted in the channel destructor.
    b. Handling of incoming bundles is queued in a global daemon event queue. Instead of the AsyncIbrDtnClient calling directly the DtnChannel callback it posts the task of running the callback with the appropriate arguments to the global I/O queue of the daemon. This way thread synchronization issues between the bundle receiving thread and the core nfd thread are avoided.
    c. DtnChannel differs from the rest of the ndn channels in that it receives all bundles for all faces, even if bundles for the same destination have appeared earlier. Other ndn channels pass connected sockets to the relevant face, which then directly receives any other bundles that may arrive at this socket, bypassing the DtnChannel.
    d. DtnChannel creates faces similarly to the way datagram channels work. The link service used is the GenericLinkService and the transport is a special DtnTransport described below. When the processBundle function is invoked by the scheduler, a face is created (or just retrieved if it already exists) and the bundle is passed on to the transport of the face.

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

5. Created DtnTransport that inherits the Transport base class. The option to use the datagram transport template class was ruled out because it is too closely tied to actual IP sockets, whereas the DtnTransport sends and receives bundles via the ibrdtn client.

    a. Created a receiveBundle function the is invoked by the DtnChannel and accepts an incoming bundle. The function reads the payload from the incoming bundle, copies the DTN payload to an NDN block element and inserts the block element into a new NDN Transport::Packet. Finally, the Transport::receive function is called with the newly created transport packet, which propagates the packet into the NDN core for processing.

    b. Implemented the virtual doSend function, which receives a Transport::Packet, creates a bundle, and copies the contents of the packet into the bundle. The bundle destination address is set to the remoteUri of the face (a face is created for each remote peer). Finally, the ibrdtn bundle is sent through an ibrdtn Client. The client is created, connected to the ibrdtn daemon, fed the bundle and closed before the function exits. Contrary to the AsyncIbrDtnClient, which is constantly running in the background, the Client created for sending the bundle has only local scope.

Android

The internal structure of the Android NFD-DTN interface is provided in Figure 12, followed by the functionality of each class or file that has been developed.

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

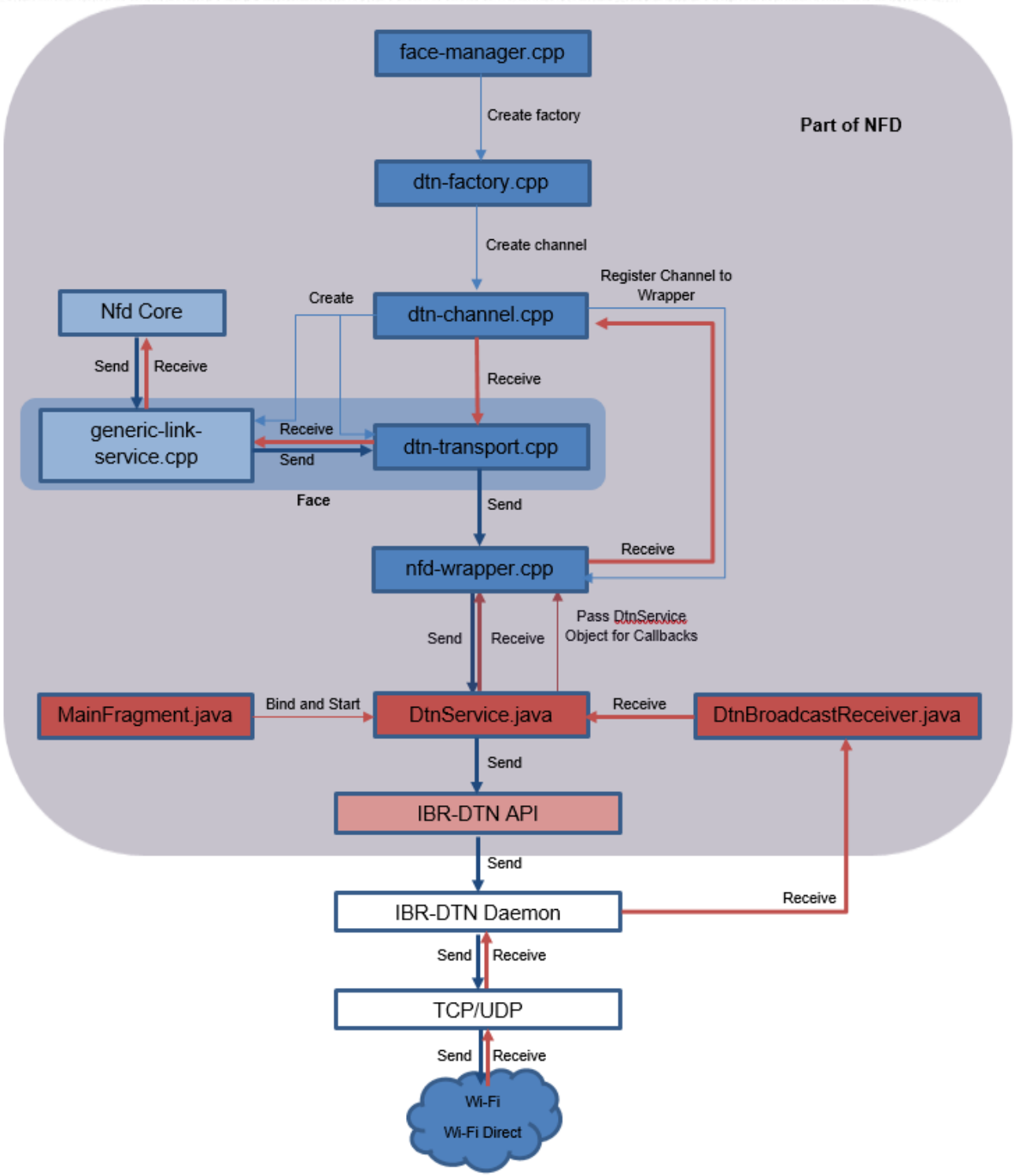

Figure 12. Internal structure for the Android DTN face

DTNService.java

The *DTNService* class is the NFD class closest to the IBR-DTN daemon. It inherits the *DtnIntentService* class of the IBR-DTN API, which supports prefix registration in IBR-DTN. Upon object creation, the *initializeNativeInterface* function is called and a *DTNService* object pointer is provided to the native part to enable data exchange with NFD.

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

DTNBroadcastReceiver.java

The DTNBroadcastReceiver class inherits the *android.content.BroadcastReceiver* class and is able to receive packets from the IBR-DTN daemon. When the latter wants to send a packet to the NFD daemon, the *onReceive* function is called, placing data in an intent[3] destined to the *DtnService* class. The *DtnService* class receives the intent using the *onHandleIntent* function, exports the content and the address of the sender and sends them to the nfd-wrapper.cpp file through the *DtnService.queueBundleJNI* function.

nfd-wrapper.cpp

The nfd-wrapper file was modified to allow the transfer of packets to and from the *DtnService* class. Upon running, pointers to the *DtnService* class and its function, *sendMessage*, are stored to improve efficiency. If an NDN packet is to be transmitted through the DTN face, the *sendToIBRDTN* function is called, which in turn calls the *DtnService.sendMessage* function.

dtn-transport.cpp

It is the "lowest" part of the DTN face and includes the *DtnTransport::doSend* and *DtnTransport::receiveBundle* functions. The former is called by the *generic-link-service.cpp* when NFD wants to transmit a packet through the DTN face. The latter is called by the *dtn-channel.cpp* file to receive incoming packets which are delivered to the *generic-link-service.cpp*.

dtn-channel.cpp

It is being created, upon NFD start, by the FaceManager through the *dtn-factory.cpp*. The *DtnChannel* class i) receives packets from the wrapper via the *queueBundle* function and then it delivers them to the NFD scheduler, so that the latter asynchronously calls the *DtnTransport::receiveBundle* function, and ii) creates the DTN face.

dtn-factory.cpp

It is being created by the FaceManager upon NFD start. It creates and administers the *DtnChannel* file.

Face-manager.cpp

[3] Intents are used by Android applications to describe future actions

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

It creates the *DtnFactory*, based on the configuration file. It also asks from dtn-channel to create a DTN face when a route is added.

4.1.2. NREP: Name-based Replication Priorities

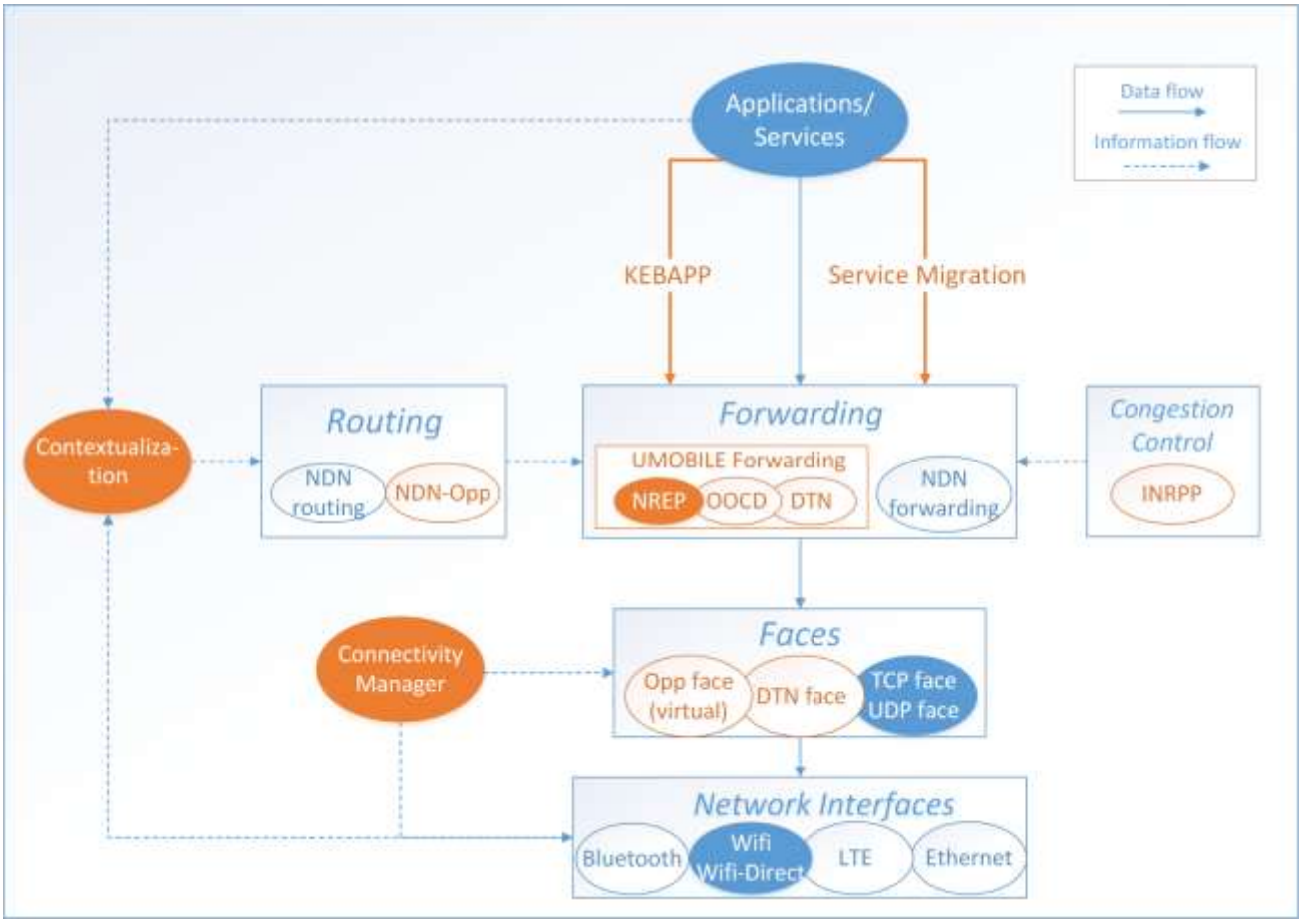

Figure 13. NREP modules

NREP has first been devised aimed at providing the best solution for emergency services transmissions using ICN opportunistic communications. These services will be supported by using replication, optimised by prioritisation rules, integrated within the information message's name to favour spreading of the most important messages. For example, in case of an emergency in a disaster area, we consider messages from first responders as more important than messages between friends. We focus on cases where the mobile network infrastructure is not available and therefore messages have to be stored, carried and forwarded by mobile devices.

In this task, we attempt to leverage the benefits of ICN in the aftermaths of a disaster, where infrastructureless delay tolerant communication becomes essential in order to deal with fragmented networks and the increase in traffic demand. Previous work done by UCL about NREP [4] use content names as the primary means for routing. However, unlike conventional ICN that is primarily designed to support name-based routing in an infrastructure-based

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

environment, NREP is designed to operate in an infrastructureless environment and focuses on name-based replication, rather than routing.

We argue for the need of a name-based forwarding/replication scheme, wherein intermediate nodes use a name associated with each message to make decisions such as whether to replicate and if so, according to what priority, or otherwise, store(-and-carry) and for how long storage should be allocated. Moreover, we discuss the need to expose other parameters such as priority, time-to-live and geographical constraints in the name or as attributes of the name.

This is done in order to help increase the efficiency of intermediate nodes to make decisions on storage and replication. However, since NREP is aimed to be developed over the UMOBILE platform and using WiFi Direct communications for Android devices, when we need to broadcast a message within a certain area, it is necessary to first establish a WiFi Direct group and broadcast the emergency message in this group, being this group formed by 2 peers or more. This connection period takes some time and adds an important latency to the communications, being up to 10 seconds in certain situations [5]. This latency cannot be omitted in mobile scenarios where contacts are short in duration time, since during that time the number of contacts around can change, losing contact with some of the users that were around. We need to prioritise those contacts with users that will have more affinity to the recipient of the message than others users.

To this end, we believe that we should take into account not only priorities between different content, but also we should have to prioritise WiFi Direct group establishments between those peers that are more suitable for their mobility history. This task considers NREP as basis as [4] to include social encounters in the connectivity manager aimed at prioritise connections with the best users to send emergency information.

Then, the task shall integrate other measures of affinity and develop NREP extensions for priorities, derived from contextual data. Examples of such measure can be aspects such as surrounding density (affinity network); priorities for the content (urgent, not so urgent); contextual input from the device (e.g., low battery). Such measures are derived from the sociability forecasting module (task 4.2, PerSense Mobile Light v2.0). PerSense Mobile Light v2.0 provides contextual data concerning roaming habits by using WiFi, data concerning social interaction, by using the WiFi 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 several UMOBILE modules, and in the case of the forwarding strategies, to the connectivity manager module.

4.1.2.1 Priorities and Namespace

In the following table, we specify an example of different priorities with its related attributes that can be defined in the NREP system and used to give priority to emergency services over other services. We think we can use this set of name prefixes as an example of different applications that can be used in the UMOBILE platform. As mentioned, these will be extended in the context of 4.3.

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

Table 1: NREP priorities example

Name-prefix	Priority	Time-to-live	Space	Authorizaiton	Recipient	Notes
SOS	High	Short	Closeby	All	First-responders	To use to ask for help
Government	High	Indefinite	All	Officials	All	To inform all of food-shelter, danger
First-responders	High	Indefinite	Depends	First-responders	All	To inform all of rescue-teams arrival
Warning	Medium	Indefinite	All	All	First-responders	First-responders verify and publish to all
Police	High	Depends	Depends	Police	Police members	To chat among themselves
Safe	Medium	Short	All	All	Public / Family	To inform others they are safe
Chat	Low	Short	All	All	Public	To chat among each other

4.1.2.2 Modules modified

- Naming scheme: The naming scheme needs to represent the content prioritisation depending on how critical is the application/service
- Content store: A new caching policy implements the prioritisation policy in order to replace first the content that is not important in terms of critical level.
- Application/services: The application services should use the specific naming scheme in order to take benefit of the proposed scheme.
- Forwarding engine: The forwarding engine includes a new forwarding strategy aimed at prioritising the critical content exchange when contacts between users occur.
- Connectivity manager: The connectivity manager has been modified to take into account the contextual information to manage and prioritise connection establishment with those users

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

socially closer to the recipient of the content or those users that will be able to better spread the content.

- Contextual information: We designed interfaces between the contextual information module (PerSense Mobile Light) and the NREP modules in order to be able to use the contextual information to improve the priority information forwarding.

The NREP forwarding mechanism is specified and developed within the Task 4.3 “Name-Based Replication Priorities” of the WP4. Deliverable D4.3 provides the details of NREP, including its modules interfaces.

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

4.1.3. Opportunistic Off-path Content Discovery

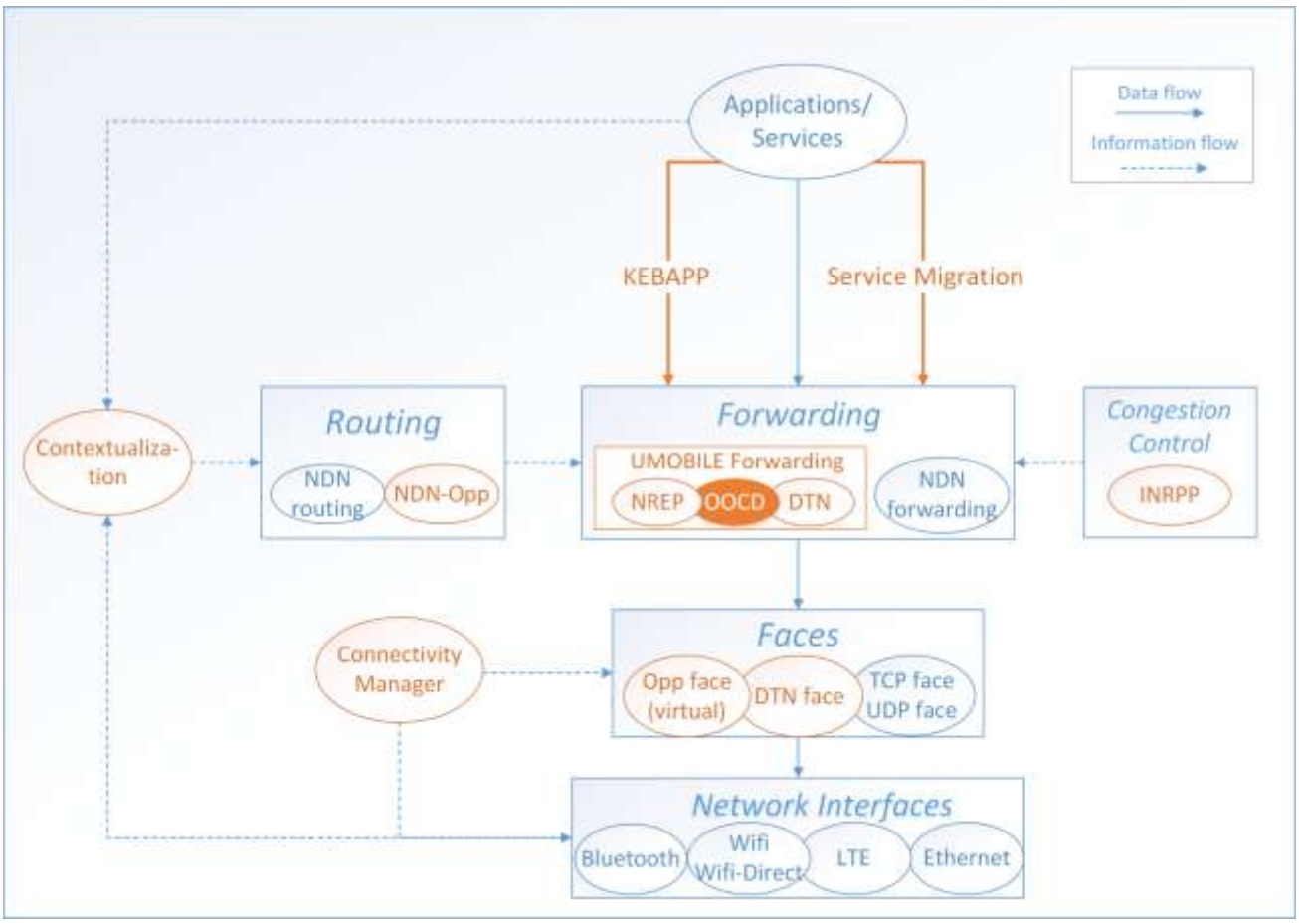

Figure 14. Opportunistic off-path content discovery modules

Data communications in fragmented networks, as can be UMOBILE networks, should not only relay on network-centric resilience schemes, as is traditionally the case, but should also take advantage of network management techniques that focus on information centric resilience. Such an information resilience scheme is useful in networks set after a disaster scenario, where the network infrastructure is only partially available (in either a temporal or spatial manner) and the reachability of the content origin is not guaranteed.

One of the main technical challenges according to the IETF ICNRG working group [6-7], regarding the usage of ICN in disaster scenarios, is to exploit network management techniques in order to enable the usage of functional parts of the infrastructure, even when these are disconnected from the rest of the network. As in [6], we also assume that parts of the network infrastructure are functional after a disaster has taken place and it is desirable to be able to continue using such components for communication as much as possible. Unfortunately, this is especially difficult for today's mobile and fixed networks which are dependent on a centralised infrastructure, mandating connectivity to central entities for communication.

The desired functionality of an ICN set after a disruptive scenario is to allow the delivery of messages between relatives and friends, enable the spreading of crucial information to citizens

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

and enable crucial content from legal authorities to reach all users in time. In such a deployment, where the content origin is not always reachable, a user can take advantage of similar interests issued by neighbouring users and their cached content in order to retrieve the requested data.

In this context, we build on the ICN paradigm and enhance the NDN architecture with extra components in order to make it resilient to network disruptions [8]. In this NDN extension, we introduce an extra Interest management routing table, which we call the "Satisfied Interest Table" (SIT) and which points to the direction of already satisfied Interests. This way, upon failure of links/nodes towards the content origin, the SIT table is redirecting Interests towards caches and end users that have recently received the requested content. We believe that network management for disruptive environments should take advantage of information-centricity, instead of focusing only on protocol-specific path recovery routing.

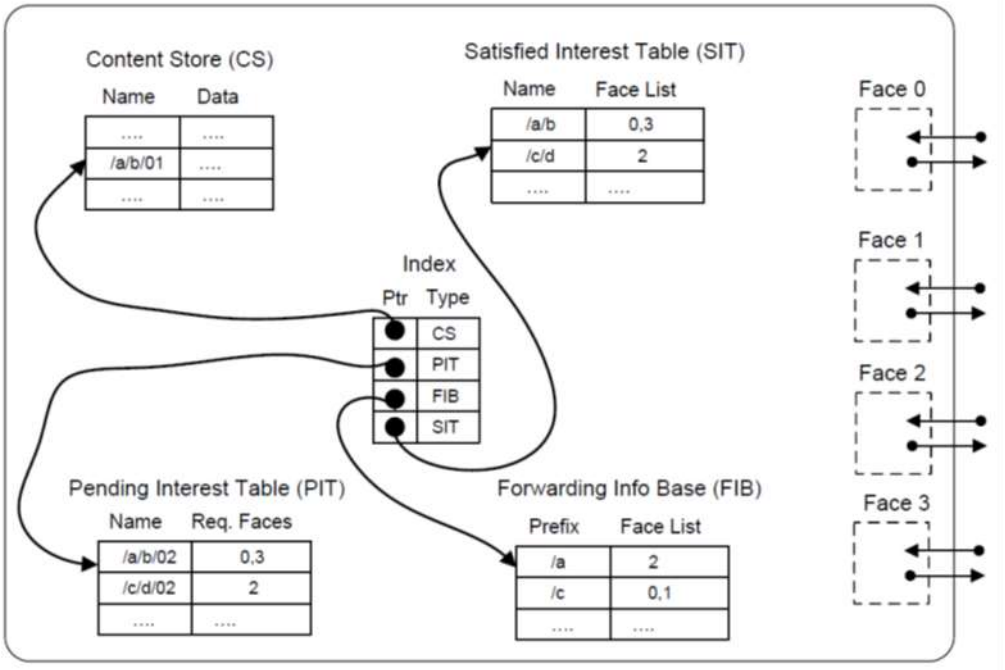

Figure 15. NDN design with the new Satisfied Interest Table (SIT)

In Figure 15, we can observe the main components of this NDN extension. As a new component of the NDN architecture, we have the aforementioned SIT table. SIT keeps track of the Data packets that are heading towards users. In the event that Interest packets cannot reach the content origin following the FIB entries, they can be forwarded based on the SIT entries towards users that issued similar interests in the past. SIT entries like PIT entries also allow for a list of outgoing faces, supporting multiple sources of data, which can be queried in parallel or sequentially depending on the chosen forwarding mechanism. Unlike a PIT entry, a SIT entry is triggered by a returning Data packet, and similarly to a PIT entry, SIT entries comprise a trail of "bread crumbs" for a matching Interest packet that leads back to users with similar satisfied interests. Here, in order to improve scalability in disruptive networks, we assume that the newly introduced SIT entries are compiled in a file/object/content item basis instead of chunk/packet IDs to speed up the opportunistic retrieval mechanism. A node removes a SIT entry associated

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

with a (directly attached) user in the event of a disconnection or when the entry expires (e.g., according to a time-to-live parameter).

We also introduce an Interest Destination flag (IDF) bit to the Interest packet in order to distinguish whether the packet is heading towards the content origin (IDF is set to zero), following a FIB entry, or is heading towards users with similar satisfied interests (IDF is set to one). In the second case (IDF is set to one), the Interest packet follows matching entries in the SIT of each passing-by router. We also introduce a Scoped-Flooding Counter (SFC) to further enhance the information retrieval capabilities of the proposed resilience scheme in some special cases that we describe in the next section. We assume here that the FIB entries of all the chunks of an item will be removed simultaneously from a router upon the "disappearance" of the content origin. This means that an Interest packet will have to visit only one network router (the one that the user is attached to) until the IDF is set to one, upon the fragmentation of the network.

4.1.3.1 Modules modified

- Forwarding strategy: The forwarding strategy has been modified in order to forward content using the three modes of operation defined by OOCD.

Opportunistic Off-path Content Discovery is aimed at enabling the usage of functional parts of the network, even when these are disconnected from the rest. This means that the local data in the fragmented network is still available, even when there is no Internet connectivity. OOCD is totally compatible with NDN and can be used with any forwarding strategy to get information from satisfied users as well as the original source. The OOCD mechanism is fully detailed in the published paper [8].

The code to simulate OOCD in the ndnSIM [16] simulator is provided in the following repository:

- https://github.com/umobileproject/OOCD

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

4.2. Contextualization agent

In what concerns data dissemination, social awareness has been in the rise in particular when considering the capability of exploiting the mobility of personal devices to reduce the need for data muling, as well as to exploit traffic locality as a way to improve service/content delivery. Of relevancy to our work are data dissemination strategies in challenged environments, where there is not always a path between source(s) and destination(s), as well as in opportunistic networks. In this type of environment, nodes are not necessarily aware of the caching possibilities, nor of the origin of content.

Several authors have developed social-aware opportunistic mechanisms for data dissemination in the context of opportunistic routing, while there is also a relevant area of work in regards to opportunistic data exchange. For instance, in the PodNet architecture users advertise the data objects that they have interest in. When two nodes meet they decide whether or not to exchange data based on the information gathered in terms of categories of interests. Contentplace builds upon this notion, adding the novelty of exchanging short summaries for the data objects they are carrying, thus contributing a decentralized dissemination solution.

UMOBILE addresses social-awareness via its context plane with the purpose of improving data dissemination, where UMOBILE takes care of the collection, resolution, and storage of the context. The context can be related to the usage, user, or to the network context. The application context can be retrieved using metadata included in data packets, in order to differentiate the characteristics of different services. This information can then be used, e.g., to adapt the routing options to the requirements of each service. The network-condition context can be retrieved and used to adapt the routing decisions to the area of action (e.g. crowded urban area or sparse network) and to the condition of each network interface (connected/disconnected, packet loss, etc). This could also differentiate the mode of operation between e.g., connected and disconnected modes. This information can be used to later decide what is the best network interface for sending a certain packet.

The UMOBILE context plane is defined by the Contextual Manager (CM, rf. to Figure 16), a UMOBILE service that runs in background on an end-user device or on an Access Point. The CM captures information concerning the device affinity network (roaming patterns and peers over time and space) as well as concerning usage habits and interests (internal device information). Such capture is either performed directly via the MAC Layer (Wi-Fi Direct, Bluetooth) as well as via native UMOBILE applications which allow the user to configure interests or other type of personal indicator preferences. For instance, an application can request a one-time configuration of categories of interests such as music, food, etc. Such metadata is passed along with the device identifier to the CM (4).

Costs derived from such contextualization are then passed, upon demand or periodically, to other UMOBILE modules to assist in different network operational aspects with the ultimate goal of achieving a more efficient data dissemination.

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

The contextualization is derived from data that is either directly captured via multiple sensors (currently, Bluetooth and Wi-Fi interfaces) as well as via external sensing applications.

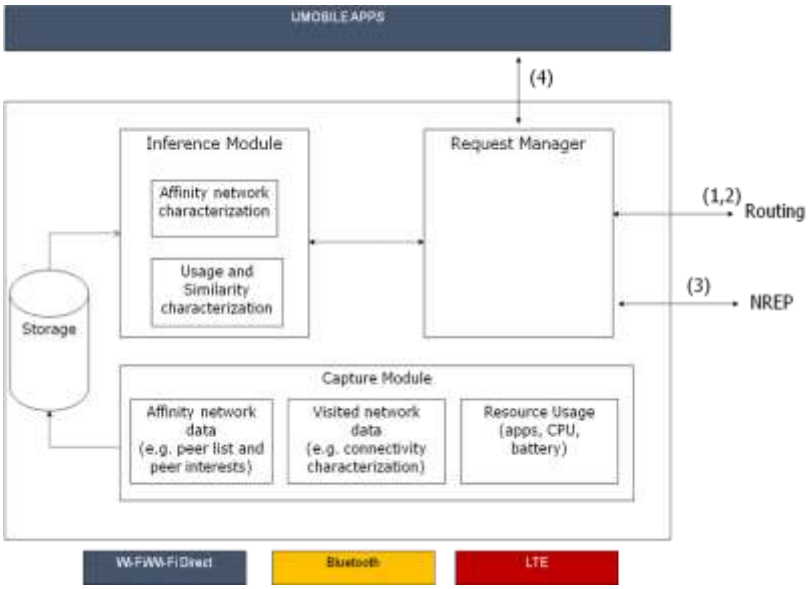

Figure 16. Contextual Manager, high-level scheme.

The CM holds four different interfaces towards other modules (two interfaces for the routing module, (1), (2); one interface for the named-based replication module, NREP (3); one interface for native applications(4)). These two interfaces allow other modules to query or to obtain information from the CM. The type of information that any CM interface provides can be categorized into two main sets: i) affinity network characterization data; ii) usage and similarity characterization data. Affinity network information concerns, i.e., peer status over time and space as well as affinities (matches) between source nodes and peers. Examples of indicators that are considered in the characterization of affinity networks and that are provided via the different CM interfaces to other UMOBILE modules are peer lists as well as interests associated with each peer, and indicators of peer status (e.g., battery) as well as of peer connectivity.:

- Peer list (bluetooth and Wi-Fi Direct) at instant t or over time window T.
- Interests associated to each peer.
- Battery status of each peer.
- Average, max, min connectivity duration over period T.
- Average. Max, min contact duration.
- Average node degree over time and space.
- Cluster distance.
- Visited networks (Aps, SSID, etc) characterization.

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

Indicators that can be provided and that concern usage and similarity characterization are built upon data collected internally (in the device). Examples of indicators that fit this category are, for instance, geo-location, as well as categories of preferred applications.:

- Preferred visited network and/or geo-location.
- Type (category) of preferred application (e.g. most used over time window T).
- Time spent per application category (e.g. per day).

Interface (1) is used by the UMOBILE routing module to request specific indicators associated with neighbouring nodes. For instance, the routing module may request a metric derived from encounter duration for each available peer, receiving in turn a list of peers with the requested parameters. Interface (2) is used to provide the UMOBILE routing module with updates concerning peers, whenever a topological change is sensed by the CM. Interface (3) provides the UMOBILE named-based replication module (NREP) with social-aware prioritisation metrics on demand or upon request. For instance, surrounding networking conditions (number of peers; average contact duration) can be combined to provide a cost on crowd density and such cost can be used to adapt not only routing decisions, as well as local data dissemination decisions (e.g. crowded urban area or sparse network).

The full specification of the contextual manager is part of deliverable D4.5. The Contextual Manager meets the following requirements of UMOBILE:

- R1: assists in understanding social trust circles; how they organize (contextual aspects) and their duration
- R4: keeps all data local; relies on network mining and not on personal raw data capture
- R10: helps in understanding the relation between location and social daily routines (roaming patterns and geographical regions)
- R18: performs seamless sensing of user context
- R20: provides feedback about networking dynamics, based on realistic social routines (roaming patterns)
- R23: allows authorized people to track the routine of registered devices
- R24: does not need an always on Internet access
- R26: allows users to manage their trust circles

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

- R15, R16: assists in inferring user interests by analysing local usage
- R18: performs seamless sensing of user context
- R24: does not need an always on Internet access
- R27: assists in matching familiar strangers' interests and assists in facilitating meetings

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

4.3. PUSH-PULL communication models

Primitive NDN supports only the pull-based communication model. However, as documented in D3.3, the UMOBILE platform needs to support both push-based and pull-based communication models to be able to serve as an implementation platform for the proposed user scenarios and their specific requirements. To cover the gap, Task 3.1 has implemented the pull-based communication model and integrated it to the UMOBILE platform. We discuss the implementation in the following sections.

In the pull-based model, a consumer (or receiver) initiates the communication. The consumer requests a piece of data that might be available from more than one producer (i.e., any node who has the matched content in the cache, original content source). In contrast, in the push-based model, a producer initiates the communication. The producer pushes a piece of data that might be delivered to one or more consumer who are interested in or subscribed to retrieve the data. A comparison between pull-based and push-based communication is illustrated in Figure 17.

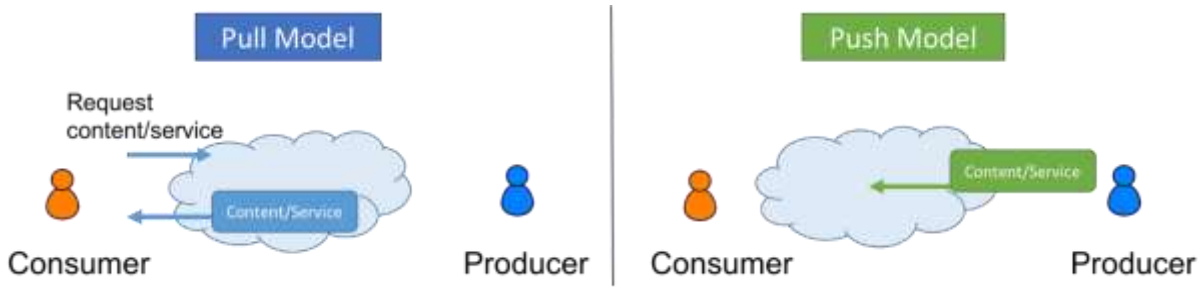

Figure 17. Push-based vs Pull-based communication models

Since the primitive NDN does not support push-based communication, a producer cannot directly send a Data message to the consumer. The reason is that NDN relies on reverse path forwarding where the forwarding Data message follows information in the PIT. However, PIT information is registered only when an Interest message is flowed to the NDN node. In other word, either the consumer or producer must send an Interest message along the path to add a new PIT entry to every intermediate node before sending the Data message. As a result, a Data message can be sent following the chain of the intermediate nodes.

In the implementation of the push-based communication model that we have done for the UMOBILE platform, push-based communication is integrated to the platform and appears as a central component of the architecture. The push-based communication services are meant to help meet the following UMOBILE network requirements as documented in the D2.2.

- R-6: UMOBILE systems MUST have an application programming interface allowing users to publish new data and subscribe/register their interests in data.
- R-8: UMOBILE systems MUST allow geo-location of users to be automatically added to contents generated by emergency applications.

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

- R-9: UMOBILE systems MUST be able to make specific types of messages available to different receivers.
- R-13: UMOBILE systems MUST provide users only with relevant information, i.e., matching user interest.
- R-17: UMOBILE systems SHOULD be energy efficient, aiming to increase their availability and reduce their operational cost.

Figure 18 illustrates our implementation of the push-based services. The functional blocks required to implement the push-based services are shown. The assumption is that the owners of the services/applications provide the communication descriptors to the Service Manager. Consequently, the Service Manager chooses the appropriate communication model (e.g., push-based or pull-based) and passes the specific description to the communication model (R-9).

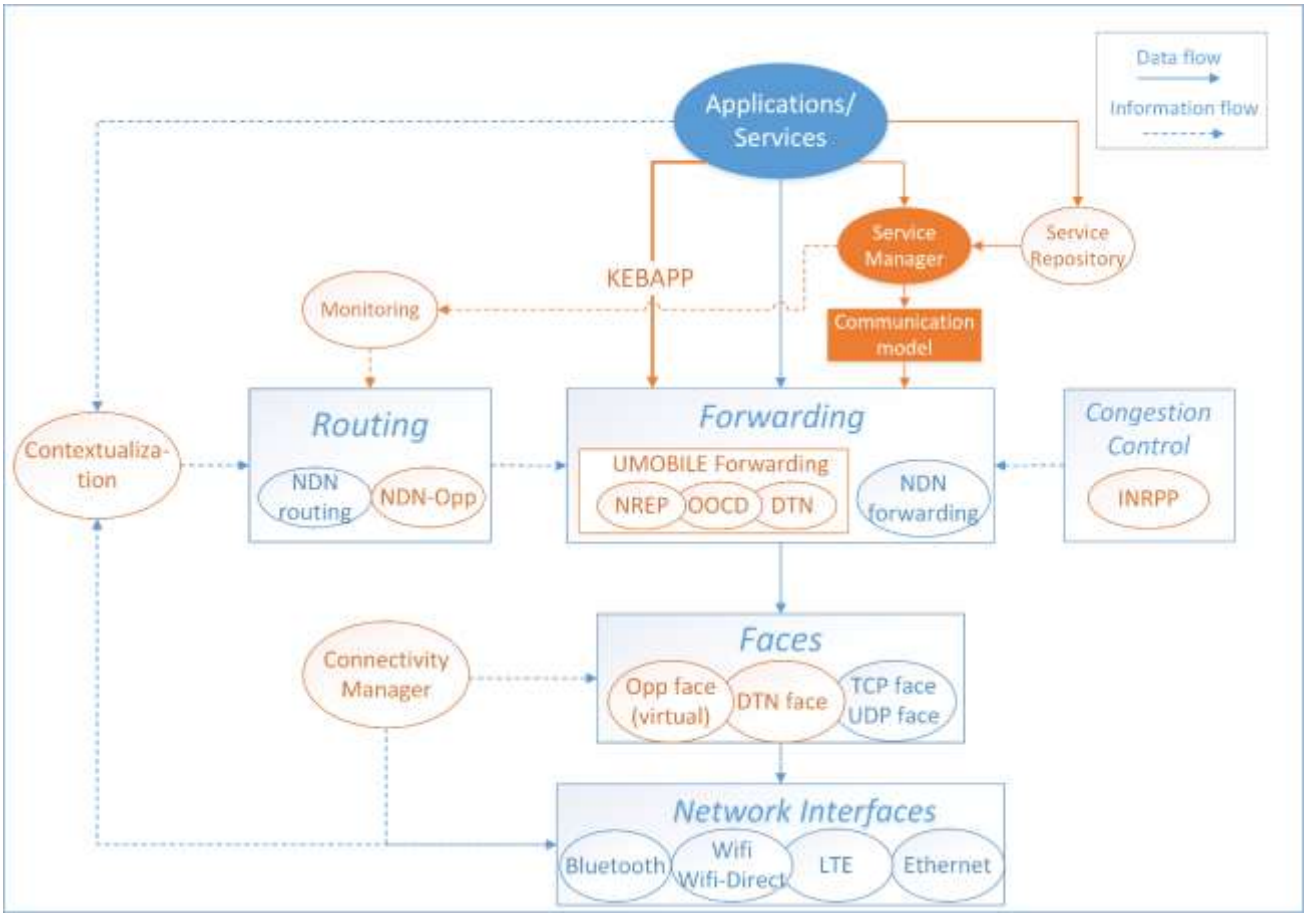

Figure 18. The functional blocks for push services (with expanded Service Migration blocks)

As mentioned in D3.1, there are four type communication models used in the UMOBILE platform. These include a typical Pull-based communication model, Push-Interest polling, Push-Interest notification and Push-Publish data dissemination. To implement these communication models,

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

we apply NDN Common Client Libraries (NDN-CCL)[4]. Currently, NDN-CCL is available in several languages including C++ (NDN-CPP), Python (PyNDN), JavaScript (NDN-JS), Java (jNDN), .NET (C#). In this section, we describe how the push and pull based communication models are developed by using Python language and PyNDN library. Without loss of generality, the same method can be applied to other languages as well. The example codes of four communication models has already been uploaded to the UMOBILE github repository (NDN-push-pull subdirectory). The examples follow producer and consumer model as mentioned in Figure 19. To run the program, open terminal and run the following command:

```
$python producer.py

$python consumer.py
```

The details of design and implementation of each model will be explained in the following subsections.

4.3.1. Pull-based communication model

The pull-based model is inherently supported by NDN. With this model, consumer initially sends an Interest message to request the desired content. The producer can respond by means of forwarding the requested content encapsulated in Data message towards the Interface following information in the PIT. This communication model satisfies the requirement R-13. Figure 19 illustrates the pull-based communication model in the NDN architecture using Interest/Data exchange.

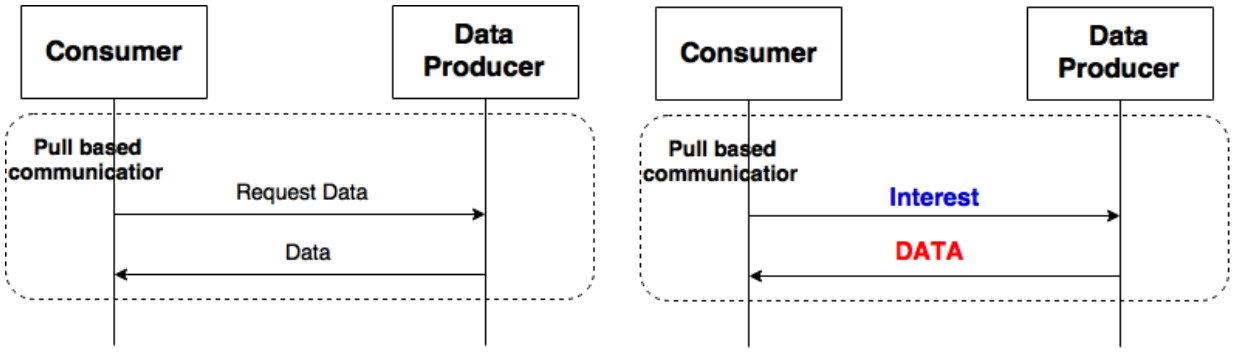

Figure 19. Pull-based communication model

In this model, the data producer initially creates a NDN face and registers name prefix. Example of creating a face is presented as follows:

```
face = Face()
```

Accordingly, the data producer uses following method to register name prefix.

[4] https://named-data.net/codebase/platform/

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

- `face.registerPrefix(namePrefix,self.onInterest, self.onRegisterFailed)`: Register name prefix to the face. *onInterest* is a callback function and invoked when the producer receives an Interest message with matching namespace. *onRegisterFailed* is called if the prefix registration process encounters an error.

The Data producer also needs to run the event loop to listen the incoming Interest messages. An example of creating a NDN face with registering name prefix can be shown in Figure 20.

```
def run(self, namespace):

# Create a connection to the local forwarder over a Unix socket
    face = Face()
    prefix = Name(namespace)

# Use the system default key chain and certificate name to sign commands.
    face.setCommandSigningInfo(self.keyChain, \
                        self.keyChain.getDefaultCertificateName())

# Also use the default certificate name to sign Data packets.
    face.registerPrefix(prefix, self.onInterest, self.onRegisterFailed)
    print "Registering prefix", prefix.toUri()

# Run the event loop for listening Interest. Use a short sleep to
    # prevent the Producer from using 100% of the CPU.
    while not self.isDone:
        face.processEvents()
        time.sleep(0.01)
```

Figure 20. Example code of creating a NDN face with registered name prefix

Consequently, the consumer creates interest object for Interest message with matching name prefix. Example of creating an Interest message is presented as follows:

```
interest = Interest(name)
    interest.setInterestLifetimeMilliseconds(4000)
    interest.setMustBeFresh(True)
```

The interestlifetime tells each NDN node to expire the Interest **N** milliseconds after it arrives (here we use 4000 ms) while the freshness selector specifies whether the NDN node's Content Store may satisfy the Interest with stale content (i.e., Data that has been held past its FreshnessPeriod).

To start the pull based communication, the consumer initially sends the Interest message by using following method:

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

- `face.expressInterest(interest, _onData, _onTimeout)`: Send the Interest and take callbacks (*_onData*) to invoke when a matching Data packet arrives and for when the Interest times out (*on_Timeout*), respectively.

In return, when the producer receives the Interest message with the matching name prefix, it replies back with Data message. Following is an example to create a Data message.

```
data = Data(interestName)
    data.setContent("Hello, " + interestName.toUri())
```

In the example above, the Data message responses the Interest message with attached "Hello" string.

4.3.2. Push-Interest polling communication model

When the consumer does not know when the data of interest will be generated and available, it can keep polling the producer by means of issuing periodic Interests. When the requested data is available, the data producer can push it in a Data message. This model can be used in energy efficient mode where a producer can turn off for a short period of time to save energy (support R-17). Once the producer is back to its ON state, it can receive the periodic Interest message and reply back immediately with the matching Data message. Two potential drawbacks of polling are that it can cause network traffic and that the freshness of the received data depends on the polling frequency. However, in some scenarios such as emergency situations where the consumer cannot be associated with any UMOBILE hotspot, this model can be useful to retrieve emergency services or contents. The Interest/Data message exchange of push-Interest polling is illustrated in Figure 21.

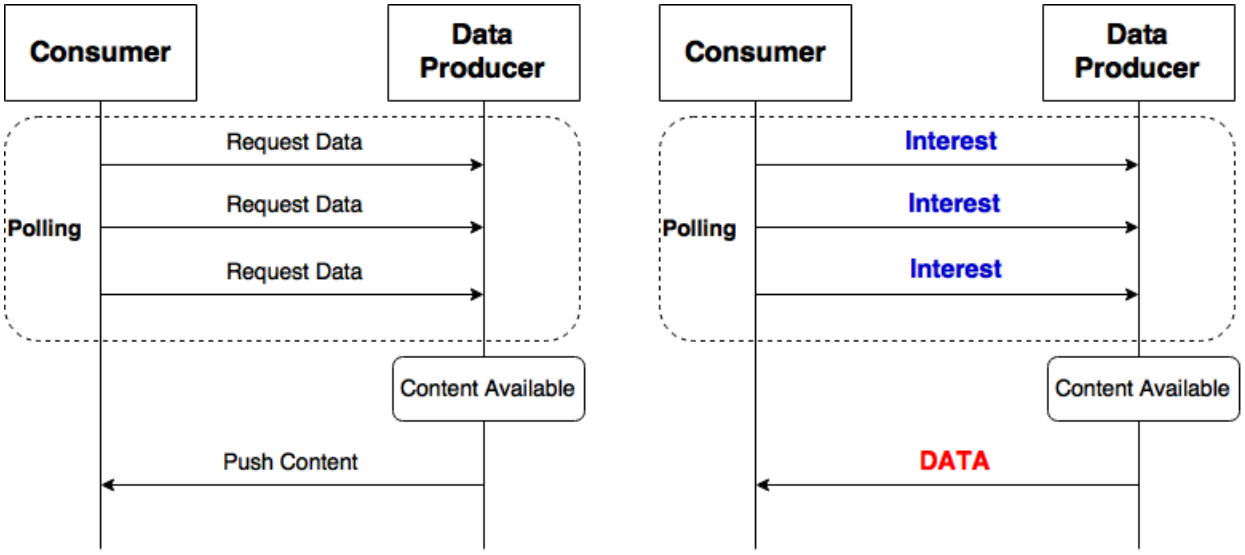

Figure 21. Push-Interest polling communication model

This model follows the same procedure as the pull based model where the data producer initially creates a NDN face and waits for coming Interest message with matching name prefix. However, the data producer can switch to the OFF state for short period. Therefore, the consumer is

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

required to run the event to send periodic Interest messages. Due to the OFF period of data producer, it is recommended that *on_Timeout* callback function should set as None. The method for sending polling Interest messages is presented as follows:

- `face.expressInterest(interest, _onData, None)`

4.3.3. Push-Interest notification communication model

In this model, the content of interest is generated as a text format i.e. as an instance message or a text notification initially sent by the data producer. This data can be directly appended to the Interest name. This method is commonly referred as *Interest notification*. Upon the reception of such an Interest, the consumer can optionally send an Ack Data message to confirm Interest reception. This model can support R-8 where the geo-location of the user can be attached to the Interest message. The push-based communication based on the Interest notification model is illustrated in Figure 22.

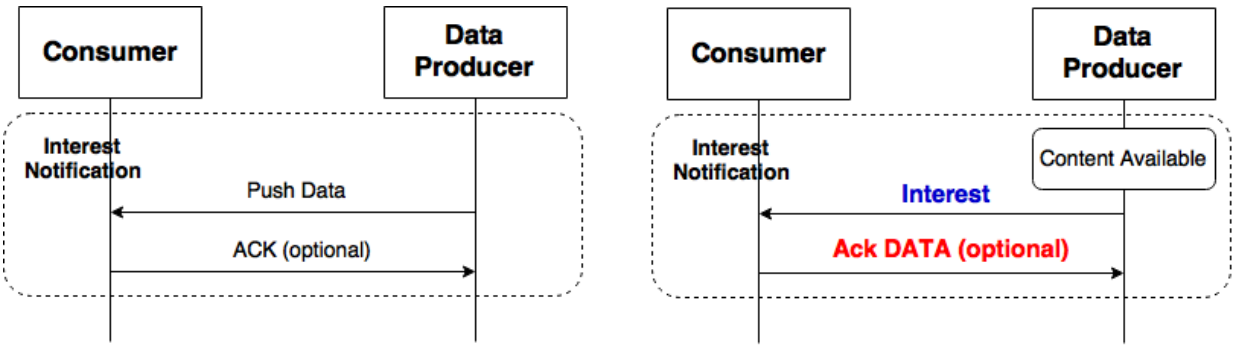

Figure 22. Push-Interest notification communication model

Unlike the pull based model, the consumer is required to create a NDN face and register name prefix before starting the communication. The consumer can use the same procedure as mentioned in Figure 22 to create a NDN face and listen the incoming Interest messages. To distinguish the type of communication (push or pull), we classify them by using the semantic name prefix. For instance, in case of Push-Interest notification, we use a following prefix *"/umobile/notification/push".*

To start the Push-Interest notification communication model, the data producer initially creates an the Interest message by appending data content to the name prefix (e.g., *"/umobile/notification/push/hello*).

The data producer can use the `face.expressInterest(interest, _onData, _onTimeout)` method as the pull based model. However, the *_onData* and *on_Timeout* callback functions can be configured as None, in case there is no Ack Data message.

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

4.3.4. Push-Publish data dissemination communication model

In this model, we follow the publish-subscribe model where data producers can publish their contents or services via Interest message to the subscribed consumer which in turn trigger a pull Interest back from the consumer to fetch data. Figure 23. illustrates the Interest/Data exchange of push-based communication with publish data dissemination under the NDN architecture. The data producer initially sends the publish message to its subscribed consumer. To distinguish the Interest message from other models, a name component, "publish" is added after the content name. Consequently, the consumer receives the publish Interest message, it discards the last component ("publish") and sends the new Interest message with the content name to request the data. In NDN, content is divided into 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 the name structure is reserved for the incremental chunk ID which is automatically generated regarding previous received data chunk.

In general, in the NDN architecture, the Data message is expected to be sent after receiving an Interest message. However, in the push-publish data dissemination model, the consumer responds to the first published Interest with a new Interest message. In this context, we use the 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 processed through the publish data dissemination scheme.

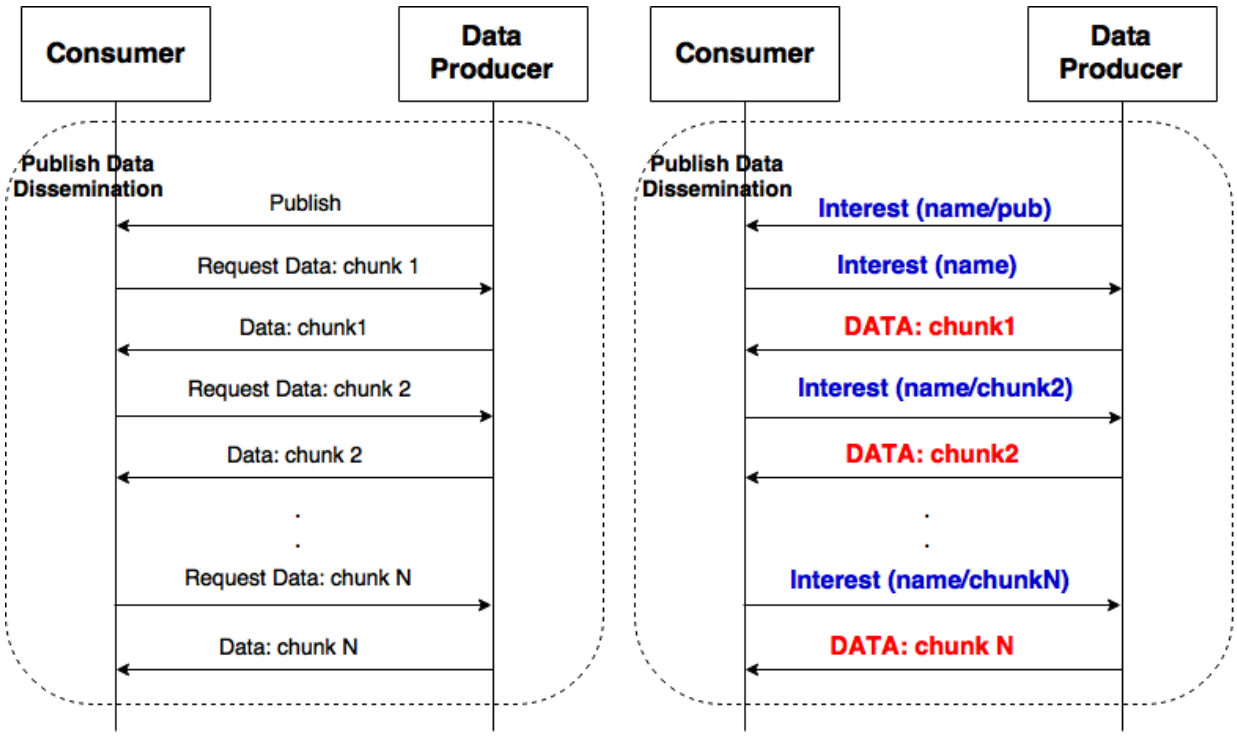

Figure 23. Push-Publish data dissemination communication model

In this model both consumer and data producer have to create a NDN face (each side) for listening the publish Interest and pull Interest messages respectively. Unlike the other models, when the consumer receives the publish Interest message, it replies back with pull Interest

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

message instead of Data message. As for the data producer, when it receives the pull Interest message from the consumer, it will reply back with subsequent Data messages as illustrated in Figure 24.

In the example code (push_PublishData_Dissemination), the data producer and consumer create a face with name prefix *'/umobile/push_PulblishData/publish'* and *'/umobile/push_PulblishData/publish'* respectively. Unlike the pull based model, the data producer initially sends a publish Interest message to the consumer by using following method:

```
face.expressInterest(interest, None, None)
```

The *_onData* and *_onTimeout* callback functions are set to none, since the data producer doesn't expect to receive Data message in return. In contrast, the data producer is waiting for a pull Interest message in order to deliver the subsequent Data messages.

When the consumer receives the publish Interest message, it responses with pull Interest message instead of issuing Data message as typical pull based model. The method to send a pull Interest message is presented as follows:

```
face.expressInterest(interest, _onData, _onTimeout)
```

Push and Pull based communication models are applied in several modules of UMOBILE architecture.

- The monitoring module requires the pull based model to fetch the monitoring data from the UMOBILE hotspot.
- The decision engine in service migration platform also requires the push - Publish data dissemination to migrate the services to the selected UMOBILE hotspots.

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

4.4. KEBAPP application sharing platform

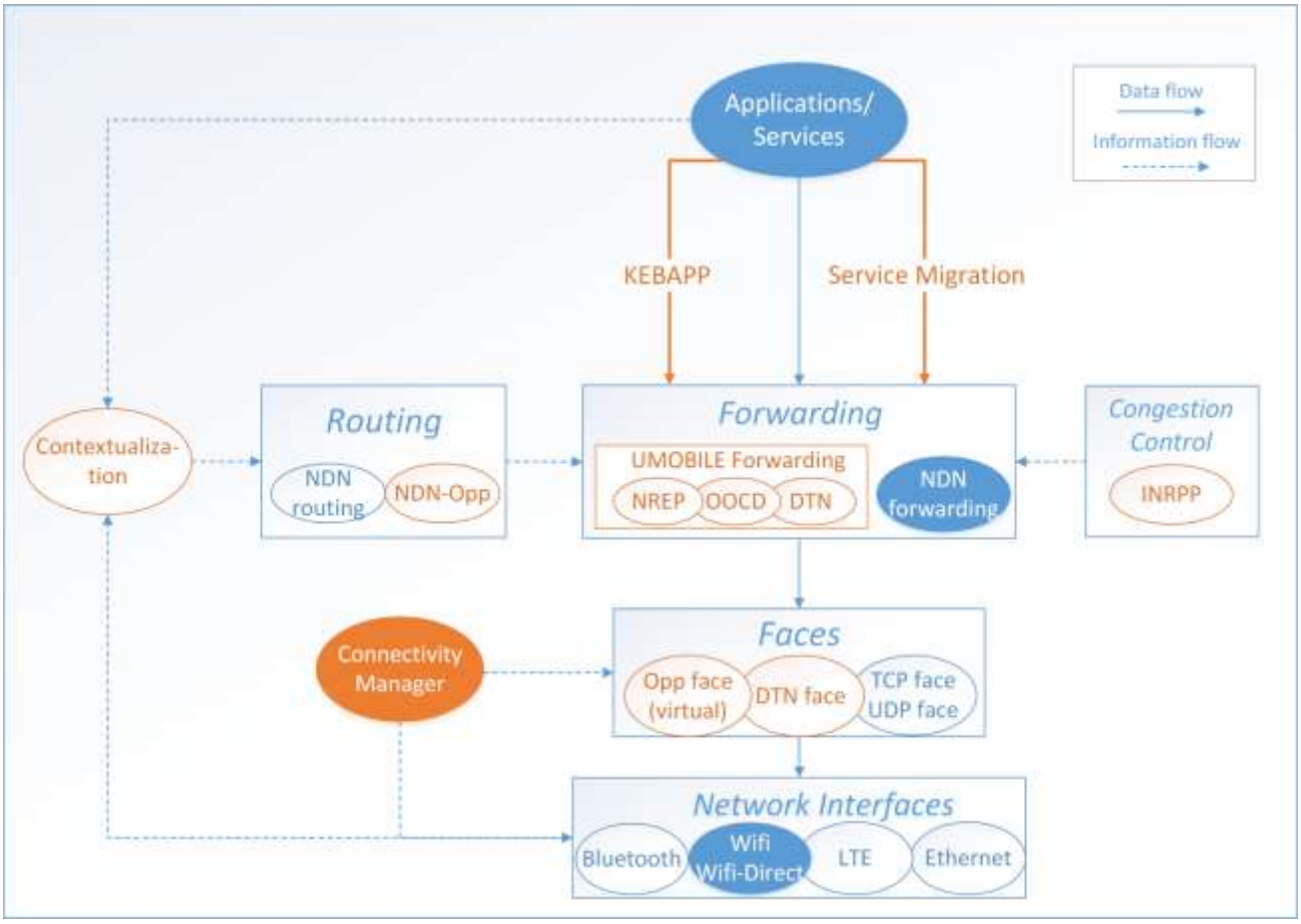

Figure 24. KEBAPP modules

Within the UMOBILE project, we present KEBAPP [9], 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. Figure 24 presents the structure of a KEBAPP-

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

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, where applications manage connectivity by defining and/or joining WiFi broadcast domains, and (iii) information-centric forwarding, extending NDN architecture.

4.4.1. 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.

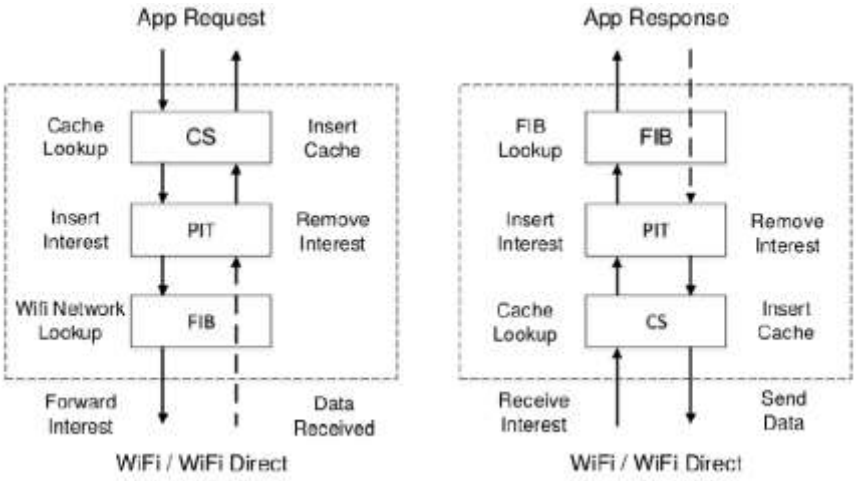

Figure 25. KEBAPP operations

Figure 25 above provides a representation of the KEBAPP packet forwarding engine. Below, we detail the operation of KEBAPP framework for an information requester:

1. The application requesting for information creates a new Interest.
2. The application looks for the information in the local CS. If the information exists locally, the data is sent to the application.

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

3. If the information is not found in the CS, the KEBAPP layer inserts an entry in the PIT table. 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 connectivity manager connects the WiFi interface to the BSSID in the FIB and broadcasts the Interest message with a corresponding time-out value.
6. 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.
7. 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 forwards 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.

4.4.2. Manual

We implemented KEBAPP on top of existing NFD implemenetation for Android [17]. 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 project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 645124

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. In order to make compatible NDN with the disruptive wireless scenario envisioned in KEBAPP, some extensions are required:

- Connectivity Manager (Faces): The Connectivity Manager is responsible of creating and destroying faces. The face manager uses the information provided by the WiFi/WiFi-Direct interfaces to create or removing faces to the multiple WiFi networks or certain local applications
- FIB Manager (FIB table): The FIB Manager is responsible of updating the FIB table. FIB manager uses the information provided by the face manager about the networks available in the vicinity and adds new entries for the KEBAPP applications available in each network.

4.4.3. Modules modified

- Naming scheme: The naming scheme use the proposed keyword-based naming scheme in order to improve scalability and provide flexibility to applications to retrieve useful information.
- Content store: The content takes into account this keyword-based naming scheme to index the data properly and forward this data when required.
- Application/services: Applications network interfaces should be modified in order to share the data with other users and retrieve data not only from central servers.
- Connectivity manager: The connectivity manager is used to create and destroy WifiDirect groups, discover other peers that are sharing the same application, and populate FIB tables with the information of the users in the vicinity.

4.4.4. Code

The KEBAPP Android code is provided in the following repository, along with a Route Finder sample application:

https://github.com/umobileproject/KEBAPP_routefinder

The Routefinder sample KEBAPP application can be found in the following link:

https://github.com/srene/routefinder

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

4.5. QoS and Congestion Control

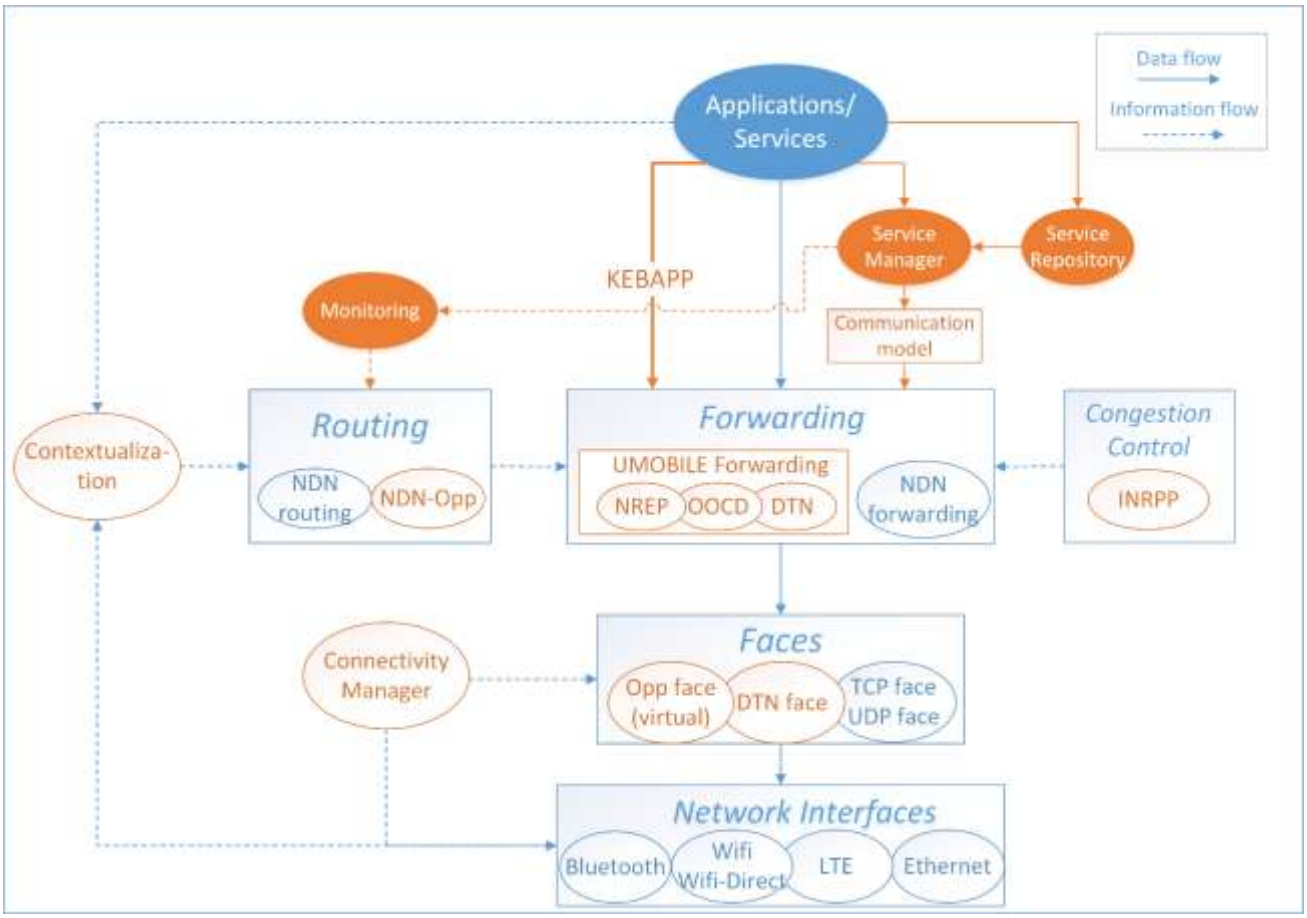

Figure 26. Service migration functional blocks (expanded)

As stipulated in Project Summary and Task 4.1, the UMOBILE architecture is expected to develop the needed mechanisms to supports different requirements of QoS. We address the issue by means of mechanisms implemented and deployed at the application level and network level that operate independently but compensate each other.

Application level QoS mechanisms exploit parameters that easy to measure and manipulate from the application level such as latency, inflicted load on servers, number of users interested in a given service, number of replicas of a service, physical location of replicas within the network, physical resources that a hotspot makes available to a service (memory, disk, CPU), precisely, to the containers (e.g., Docker containers) hosting the instances of the service, and so on. On the other hand, **network level QoS mechanisms** are based mainly on network traffic engineering and exploit parameters that are easy to measure and manipulate at network level such as link throughput, link outages and traffic priotization to favour traffic belonging to services of premium classes. The discussion of this section is focused only on application level QoS mechanisms. Note that application level QoS mechanisms rely on both, the optimisation of the usage of available resources and the possibility of deploying additional ones such as additional virtual machines, for example, Docker containers.

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

4.5.1. Service migration

In this section we discuss the design and implementation of a service migration framework that we are in the process of implementing as a measure to address QoS requirements as expected within the work to be carried out by Task 4.1. We also explain how the features provided by the service migration framework help satisfy the system and network requirements stipulated in the D2.2 Deliverable (*System and Network Requirements Specifications).*

4.5.1.1 System and network requirements

The motivation for the implementation of the service migration framework is to enable the UMOBILE platform to satisfy some of the system and network requirements stipulated in the D2.2 deliverable. The following requirements are those related to service migration. By service migration we mean the service migration framework that we are in the process of implementing to perform service migration and replication as explained in subsection 4.5.1.2.

- R-6 UMOBILE systems MUST be able to ensure data reliability and availability (e.g. taking into account data usefulness - time to live; manage duplicated pieces of information) among a set of distributed surrogates.
- R-8 UMOBILE systems MUST be able pre-fetch data in order to improve service performance.
- R-10 UMOBILE systems MUST be able to deliver information within geographic regions and time frames that are relevant to different types of data.
- R-11 UMOBILE systems MUST provide users only with relevant information, i.e., matching user interest.
- R-13 UMOBILE systems MUST be able to provide the services to the end users when there is no Internet connectivity.
- R-16 UMOBILE system SHOULD be able to pre-fetch data based on user interests (e.g., parking places near recommended art gallery) and behaviour (e.g. mobility patterns), in order to reduce delays in data delivery.
- R-19 UMOBILE systems SHOULD provide information about the network status (e.g. network diameter, average path length, link bandwidth, network delay) is order to allow authorities to take corrective measures (e.g. deploy UAV infrastructure).
- R-20 UMOBILE systems SHOULD be able to create opportunistic communication infrastructures, instantaneously deployed using UAVs.
- R-22 UMOBILE systems SHOULD be able to provide local services when the system cannot connect to the Internet.

The relationship between the requirements listed above and service migration is explained in the following lines:

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

- R-6: Data reliability and availability can be achieved by means of migrating services in anticipation of potential network outages, that is, upon detection of threatening problems.

- R-8,R-16: The service migration platform implements service pre-fetching mechanism aimed at improving service performance.

- R-10: Service migration accounts for geographical locations in its service migration strategies, for instance, a service is migrated to the geographical region where users have expressed demand for the service.

- R-11: A service is migrated only when the service migration engine estimates that there is enough user interests to justify the migration costs.

- R-13, R-22: Service are migrated as close as possible to the end user (normally to the network edge) and as a measure to mitigate connection problems including slow connectivity and or no connectivity at all.

- R-19: The service migration framework will include monitoring mechanisms for collecting metrics (topology, latency, link bandwidth) that help determine the current status of the network.

- R-20: This requirement is to some extend related to the service migration framework in that, the latter is based on opportunistic service migration.

4.5.1.2 Service migration overview

The service migration platform is a core components of the UMOBILE architecture. Service migration assumes a network provider-centric model where the provider is in full control of the communication infrastructure. The service migration platform takes advantage of three fundamental concepts: Light weight virtualization technology, Opportunistic service deployment and Edge network computing. Our architecture is underpinned by lightweight dockerised services[5], built as self-contained VMs coupled with a semantic naming scheme. These services can be seamlessly deployed and executed across all compatible devices within the UMOBILE network (e.g., a hotspot). We take advantage of caching and name-based routing to allow users to access nearby copies of the service. Furthermore, thanks to the semantic naming scheme, a user can request a service that matches certain criteria.

In brief, the central idea is to deploy services (either reactively or proactively) that can be instantiated from Dockerised imaged at hosts located at the edge of the network and with

[5] https://www.docker.com

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

enough resources to run them to ensure that the services will satisfy their QoS constraints. The main components of the service migration platform are shown in Figure 27.

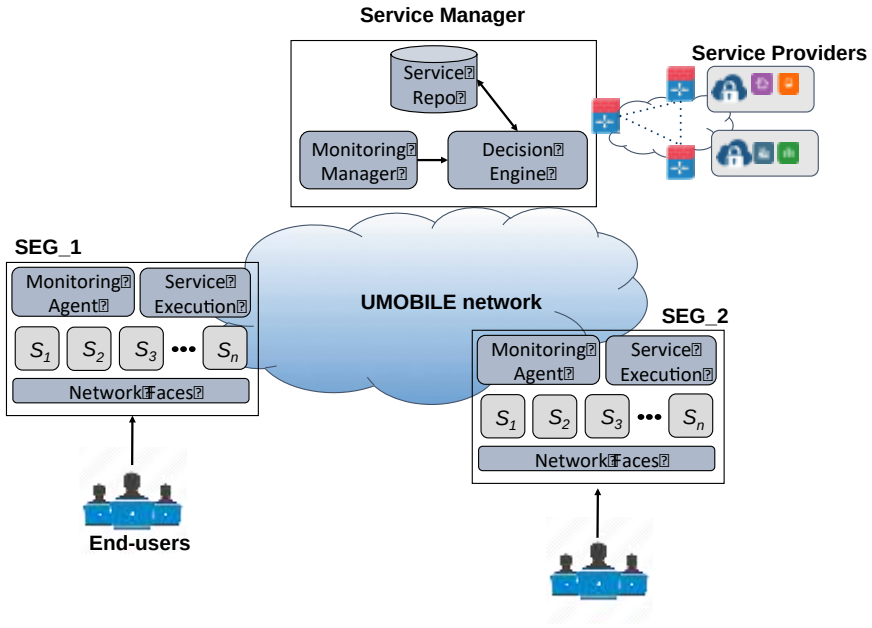

Figure 27. A view of UMOBILE platform deployment with focus on Service migration

- **UMOBILE Network:** The network is a UMOBILE network is the sense that the routers (four of them as shown) are deployed with the UMOBILE platform and provided with disks that they can use for content caching.

- **Service Providers:** A service provider is a business entity that is responsible for delivering services. In the UMOBILE project we assume a provider-centric business model where the service provider is in full control of the communication infrastructure (i.e., routers, access point, IoT sensors) and several source constrained edge nodes such as single board computers. The rationality behind this decision is that this business model is increasingly gaining acceptance in the service provisioning market.

- **End-users:** An end-user (represented by smart phones, laptop) is a person in possession of a mobile device with wireless facilities and interested in accessing services provided by the ISP provider.

- **Application/Services:** The Application (also called Services) are compressed dockerised images produced by content providers and delegated to the Service Manager for deployment under the observance of QoS requirements. They are stored in the Service Repo shown in Figure 27.

- **Service Execution Gateway (SEG):** A SEG (also called an edge node) is a single board computer such as a Raspberry Pi (RPi) or Internet home router with storage, CPU and software facilities for hosting (upon request of the service controller) the execution of virtualised services. Two of them are shown in the figure (SEG_1 and SEG_2) but there

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

might be an arbitrary large number of them. SEGs are integrated with *Monitoring Agent* and *Service Execution* classes implemented in software.

  - **Monitoring Agent:** is a piece of software that is responsible for measuring the current status of resources and the current demand imposed on the services. By resources we mean, the actual hardware used to realize the SEGs (implemented by Raspberry Pis) and the Docker containers instantiated in the hardware. As explained in details in Section 4.3.1.2, the monitored data is formatted as json objects.

  - **Service Execution:** is a piece of software that we have implemented to enable SEGs to instantiate containers automatically upon receiving requests from the Service Manager.

- **Service Manager:** is a piece of software that responsible for making informative decisions on the deployment of services. As such, the Service Manager is the core of the service migration platform. It includes three software components, namely, *Service Repo, Monitoring Manager and Decision Engine.*

  - **Decision Engine:** implements the logic for opportunistic deployment of instances of services to meet QoS requirements. It operates on the basis of the QoS requirements of individual services, service demand expressed by end users, current status of resources (network and SEGs) and algorithms that help determine where and when to deploy a requested service.

  - **Monitoring Manager:** is responsible for placing pull requests against the *Monitor Agent* deployed in each edge node to collect information about the current status of their resources and the current demand imposed on the service. As shown in Figure 28, the *Monitoring Manager* provides an interface to the Decision Engine to retrieve json objects with monitored data that includes information of all SEG nodes.

  - **Service Repo:** is a repository where dockerized compressed images of the services are stored augmented with specification about their QoS requirements.

4.5.1.3 Implementation of service migration

We have developed the service migration platform in Python following an object-oriented design. Figure 28 illustrates the main classes and interfaces of the service migration platform. As shown in the figure, the components use json files to communicate with each other. In principle, other communication formats can be used such as XML, we have favoured json because it is simple, well documented and widely accepted in current technology. The service migration is implemented based on dockerised technology. In our implementation, we use x86 machine (e.g., laptop, desktop or sever) to run as the service manager. The installation of docker is available in

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

the official docker site[6]. As for the SEG, we use raspberry Pi as a platform for implementation which has different architecture and cannot use the same instruction as x86 machine. Therefore, we have prepared an image for raspberry Pi integrated with docker which is already available in the UMOBILE github. Apart from docker installation, following are the prerequisite packages required to be installed on both service manager and SEG machines.

```
% sudo apt-get install build-essential libssl-dev libffi-dev python-dev python-pip
% sudo easy_install pyndn
% pip install docker
```

Notice that, we use symbol "_" and italic font style to express the classes of the objects. For instance, the Monitoring Manager is presented as "*Monitoring_Manager*"

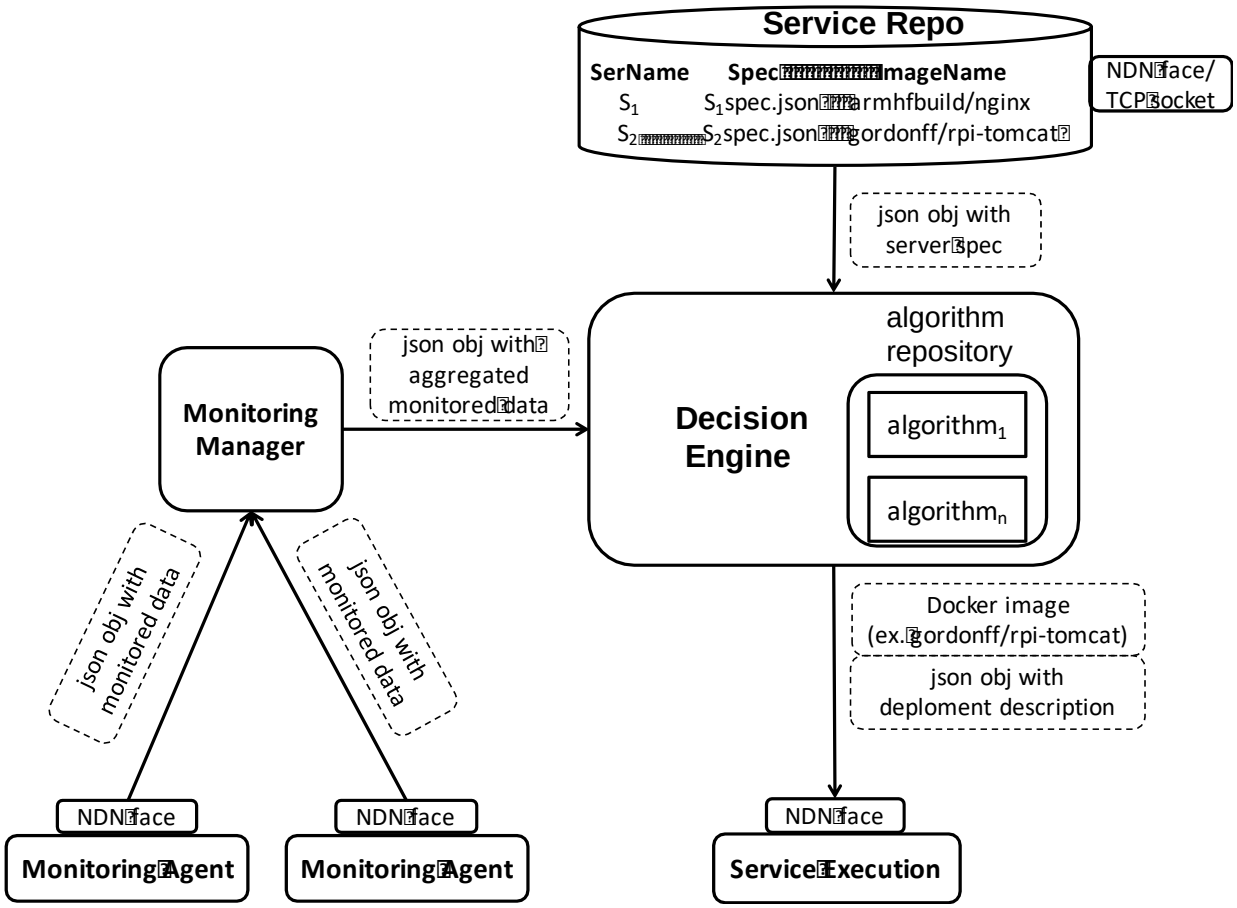

Figure 28. Diagram of the main classes and interfaces of the Service migration platform

4.5.1.3.1 Service Manager

The Service Manager is the master module of the Service Migration platform and is composed of the *Service_Repo, Monitoring_Manager* and *Decision_Engine* classes. Its responsibility is to coordinate the operation its components modules in opportunistic service deployment. The service manager also offers an interface to the end user, namely, to the service provider. Such

[6] https://docs.docker.com/engine/installation/

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

interface is accessible from a conventional Linux shell. The current implementation does not offer yet an API to access the Service programmatically. Neither does it offer the Service Provider a sophisticated GUI (Graphical user Interface) to help him or her configure, upload and retrieve information from the component modules; for example, to populate to Service repo or display the list of images stored in its disk. These and other operations can be incrementally added to the interface. For the time being, our effort is focused on the core operations that the Service Manager needs to support to proof the concept. To start the Service Manager module, issue the following command from a conventional terminal (for example, a Linux shell).

```
% python service_manager.py
```

4.5.1.3.2 Monitoring Manager

The Monitoring Manager is responsible for aggregating the monitored data from all the Monitoring Agents and making it available to the *Decision Engine*. We implemented the *Monitoring_Manager* as a python class. A *Monitoring_Manager* object is created by the Service Manager. This class is controlled by threading while it periodically sends an Interest message (pull) to a specific SEG. Example of the usage of the class is presented as follows:

```
Monitoring_Manager (thread-ID, "thread-name", namePrefix, monitoring_frequency)
```

thread-ID (numeric number, e.g., 1) and *thread-name* (e.g., "Monitoring-Thread-SEG_1" arguments are used to control the threading operation such as terminate or restart process. The *monitoring_frequency* is a periodic time period that the *Monitoring Manager* sends an Interest message to the SEG. The *namePrefix* parameter is a name for each Interest message. An example of *namePrefix* that monitoring manager sends to SEG_1 is *'/umobile/monitoring/SEG_1/'.*

The following are the main methods implemented by the *Monitoring_Manager* class.

- `_sendNextInterest(name):` Send an Interest message to SEG. The input parameter *name* is assigned by *MonitoringManager.* It is also used to set the name of Interest message.
- `_onData(interest, data):` Process the Data message sent by *MonitoringAgent.* The input parameter *interest* is a pending Interest entry in PIT which has the same name as Data message. The argument *data* is receiving Data message. The monitoring data (e.g., status-SEG_1.json) encapsulated in the Data message will be extracted and recorded in the local subdirectory called Monitoring_DB.

4.5.1.3.3 Decision Engine

The *Decision Engine* is responsible for making decisions on deployment of service instances. The *Decision_Engine* is implemented as a python class. A *Decision_Engine* object is created by the *Service Manager*. The *Decision_Engine* is controlled by the threading where it creates a face to response to Interest message pulling the service images. For example, to create an object of *Decision_Engine* class, we proceed as follows:

```
Decision_Engine(thread-ID, "thread-name", namePrefix)
```

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

thread-ID (numeric number, e.g., 1) and *thread-name* (e.g., "Thread-DecisionEngine") arguments are used to control the threading operation such as terminate or restart process. The *Decision_Engine* also creates a NDN face and registers it with a given *namePrefix*. An example of *namePrefix* is *'/umobile/service_deployment_pull/'.*

The following are the main methods implemented in *Decision_Engine* class:

- `face.registerPrefix(namePrefix,onInterest_PullService, onRegisterFailed):` Create a NDN face with given *namePrefix*. *onInterest_PullService* is a callback function executed when *Decision_Engine* receives an Interest message matched with the given *namePrefix* (i.e., *'/umobile/service_deployment_pull/*). *onRegisterFailed* is a response function when Decision Engine fails to create a NDN face, it returns an exception error message on the terminal.
- `face.expressInterest(interest, None, None):` Send a Pull Interest message to the SEG. This method supports Push based Publish data dissemination model as introduced in [1, 2].
- `onInterest_PullService(self, prefix, interest, face, interestFilterId, filter):` Process an Interest message requesting for a service. The method sends a requested service image back to the SEG along the reverse path forwarding.

The Decision Engine has access to a repository of algorithms implemented to make decisions on deployment of instances of services. These algorithms vary in the QoS requirements that they try to meet as well as on their accuracy and complexity. For instance, when a SEG node shows signs of exhaustion as a result of a large number of requests placed against one of its services, the Decision Engine reacts to ease the situation before it is noticed by the end-users. Several strategies can be taken to address the problem. The service migration platform addresses it by means of deploying more instances of the service under stress. *de* class represents the algorithm repository that contains a set of algorithms for *Decision Engine.* In the following example, a *de* object is created by *Decision_Engine.* The input "monitoring path" is a path to Monitoring_DB subdirectory containing fresh monitoring data of the SEGs.

```
de("monitoring path")
```

The following are the main methods implemented in *de.*

- `de.selectHost_to_deploy_firstInstance(json_lst_dict, json_server_Spec)` : Select a suitable SEG for deploying a the first instance of a service. This method returns a string representing the ID of a SEG or None if no SEG is found. *json_lst_dict* is a list of json dictionaries, *json_server_Spec* is a json file with the specification of the server to deploy. The output of the algorithm is the name of the selected SEG.

  The list of json dictionaries (json_lst_dict) deserves further discussion. As shown in the figure, we deploy a Monitoring Agent within each Hotspot that is responsible for collecting metrics about the current status of its hotspot (two Monitoring Agents are shown in the figures 27 and 28). Each Monitoring Agent collects its metrics, places it is a json dictionary (see discussion of the Monitoring Agent, below) and makes the json

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

dictionary available to the Monitoring Manager. The Monitoring Manager is responsible for aggregating all the json dictionaries (one from each Hotspot) and making the aggregation available to the Decision Engine. The json_lst_dict input is the aggregation. We have implemented it as a conventional Python list that includes all the individual json dictionaries.

Regarding the json_server_Spec, it is worth clarifying that it corresponds to the Spec files shown in Service Repo of Figure 28 (see S1spec.jon and S2spec.json). It might be helpful to recap that as shown in the figure, each service is associated to a service specification json file that describes the features of the service that are relevant for its deployment. For example, the size of the service image and its class is stipulated in this specification files.

- `de.try_localReplication_of_additionalInstance(json_lst_dict, json_server_Spec, host, cpuLoadThreshold):` Determine if a given hotspot has enough resources to host the deployment of an additional instance of a service. The method returns a boolean value (yes or no) and takes for arguments. As in `de.selectHost_to_deploy_firstInstance,` json_list of_dict is a list of json dictionaries and json_server_Spec is the specification of the service to be deployed. Host is the ID of the host that we are examining and cpuLoadThreshold is the threshold value that the method uses to determine if the host is exhausted or vigorous. Note that the current implementation of this method is very basic as it only accounts for the status of the cpuload of the host. A more realist implementation would account for other parameters that determine exhaustion such as current memory usage, number of containers running in the host, and so on.
- `Ancillary functions:` The algorithms of the decision engine need the support of several ancillary methods and functions. To explain the point, we will discuss some examples of ancillary functions. In these functions, json_lst_dict is a json list of dictionaries as described in `selectHost_to_deploy_firstInstance` while host is the ID of the hotspot of interest.
    - get_num_of_containers_of_pi(json_lst_dict, host): A Pi used for realizing a hotspot can host the execution of zero or more containers. This function produces the number of containers (an integer) currently running in a Pi.
    - get_pis_with_cpuLoad(json_lst_dict): The Pi can experience different levels of cpu load. This function produces the current cpu load (a float number) of all the Pis and sort them is increasing g order.
    - get_pi_cpuLoad(json_lst_dict, host): This function produces the current cpu load (a float number) of a given Pi.
    - get_pis_with_min_cpuLoad(json_lst_dict): To decide on deployment of containers, it is useful to identify the Pi that is currently less loaded. This function produces the ID of such a Pi. Note that the Pi is not necessarily unique, there might be one or more of them experiencing the same load. The function selects one of them arbitrarily.

The actual code of all these methods is available from Github.

4.5.1.3.4 Service Repository

Service Repository is a repository where dockerized compressed images (s_i) of the services are

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

stored augmented with specification about their QoS requirements. The *Service_Repository* is implemented as a python class. The *Service Manager* creates an object of that class. The *Service_Repository* is controlled by the threading where it creates a NDN face and conventional TCP socket to communicate with service providers. An example of the usage of the class is presented as follows:

```
ServiceRepo(thread-ID, "thread-name", namePrefix, port_num)
```

thread-ID (numeric number, e.g., 1) and *thread-name* (e.g., " Thread-ServiceRepo") arguments are used to control the threading operation such as terminate or restart process. The *Service_Repository* also creates a NDN face and registered it with a given *namePrefix*. An example of *namePrefix* can be expressed as *'/umobile/service_repo/'.* The service providers will upload its service specification in json format and dockerized service image through this face. However, our implementation also creates a conventional TCP socket to communicate with typical service providers. The parameter *port_num* is used to setup the TCP socket.

To describe the specification of services we have taken a pragmatic and simple approach--- use json notation. An example of a service specification of service S1 is shown in Figure 29

```
S1_spec=  {'par':{
          'serviceName':  'S1',
          'imageName':
'armhfbuild/nginx',
          'imageSize':    '368',
          'maxUsers':     '50',
          'startUpTime':  '5'
          }
```

Figure 29. An example of service description of service S1

The *par* key contains a set of (key, value) pairs related to the features of the service. For example, (imageSize, 368) indicates that the size of the image is 368 MB. Likewise, the pair (maxUsers, 50) indicates that an instance of the service can handle up to 50 concurrent users. Finally, (startUpTime, 5) indicates that it takes 5 seconds to start up an instance of the server. The *QoS* key contains a set of (key, value) pairs that stipulate the QoS requirements. In this order, S1 is expected to support at least 100 concurrent users, be available 99.99% of the time and respond to request within 5 seconds.

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

4.5.1.3.5 Service Execution Gateway (SEG)

A SEG (also called an edge node) is a single board computer such as a Raspberry Pi (RPi) or Internet home router with storage, CPU and software facilities for hosting (upon request of the service controller) the execution of virtualised services. Two of them are shown in the figure (SEG_1 and SEG_2) but there might be an arbitrary large number of them. SEGs are deployed with *Monitoring_Agent* and *Service_Execution* classes. To start the Service Manager module, use the following command in terminal:

To start the SEG, issue the following command from a terminal:

```
% python seg.py

Enter SEG-ID (e.g., SEG_1) SEG_1
```

The script is required to add input for SEG-ID (e.g., SEG_1).

4.5.1.3.6 Monitoring Agent

The *Monitoring_Agent* class is responsible for collecting monitored data from its underlying hardware and running containers. Upon request from the *Monitoring_Manager*, a *Monitoring_Agen*t provides the latest status of the resources of its SEG node and of the containers (zero or more) currently deployed. *Monitoring_Agent* sends several *json obj with* monitored data objects to the *Monitoring_Manager*, all depending on the frequency of metric collection. The meaning of the keys included in the object are as follows:

- **SEG node:** configuration and current status of the resources of the edge node.
    - *PiID*: Identification of the SEG node.
    - *PiIP*: IP address assigned to the SEG node.
    - *hardResources*: hardware configuration of the SEG node.
    - *softResources*: software configuration of the SEG node.
    - *resourceUsage*: current status of the consumables resources of the SEG node. For example, cpuLoad is the load currently inflicted on the cpu of the SEG node.
- **Containers running in the SEG node:** configuration and current status of the resources

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

of the containers (zero or more) currently running in the SEG node.

- *id*: identification of the container
- *cpuUsage*: amount of cpu currently consumed by the container.
- *memUsage*: amount of memory currently consumed by the container.
- *name*: the name of the service (application) hosted in the container.
- *status*: amount of time spent by the container running.
- *image*: name of the image used for creating the service.
- *port host*: port number allocated to the service.
- *port container*: port number allocated to the container.

The *Monitoring_Agent* is implemented as a python class. An object of that class is created by the SEG. The *Monitoring_Agent* object is controlled by the threading while it creates a face for Interest request from *Monitoring_Manager*. Example of the usage of the class is presented as follows:

```
Monitoring_Agent(thread-ID, "thread-name", "seg_ID", namePrefix)
```

thread-ID (numeric number, e.g., 1) and *thread-name* (e.g., "Thread-Monitoring") arguments are used to control the threading operation such as terminate or restart process. *seg_ID* is unique ID of target SEG (i.e., SEG_1). The *namePrefix* parameter is used to publish its data content mapping with the NDN face. An example of *namePrefix* can be expressed as *'/umobile/monitoring/SEG_1/'.*

The following are the main methods implemented by *Monitoring_Agent* class.

- `face.registerPrefix(namePrefix, onInterest, onRegisterFailed)`: Create a NDN face with given *namePrefix. onInterest* is a response method when *Monitoring_Agent* receives an Interest message matched with the given *namePrefix. onRegisterFailed* is a response function when Decision Engine fails to create a NDN face, it returns an exception error message on the terminal.
- `onInterest(namePrefix, interest, face, interestFilterId, filter)`: Process an Interest message requesting for monitoring data. The method sends a json file back to the *Monitoring_Manager* along the reverse path forwarding.

A *termopi* object is created by *Monitoring_Agent* to measure real-time resource consumption of the underlying hardware and running containers. An example of the usage of *termopi* is presented below:

```
termopi()
```

The following are the main methods implemented by *termopi* class:

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

- `create_jsonfile_with_pi_status(file_path)`: Measure real time values of resources consumption of underlying hardware (i.e., raspberry Pi) and running containers. The method returns monitored data formatted as a json file.

The monitored data is formatted as json objects. An example of actual monitored data collected from SEG_1 is presented in Figure 30.

```
{"softResources":{"OS": "Linux"},
"hardResources": {"mem": "1 GB", "disk": "16 GB", "cpu":
                            "A 1.2GHz quad-core ARMv8 CPU"},
"resourceUsage": {"cpuLoad": "0.04","memUsage": "14",
                  "cpuUsage": "24.31"},
"PiID": "SEG_1",
```

Figure 30. Monitoring data collected by termopi class

The *resourceUsage* key contains the values of current CPU load, memory usage and CPU usage. Also the object informs that SEG 1 is currently running a single container that has been running for 3 hours, has the name */adisorn*, has used 14% of memory and 24.31% of CPU. This information is regularly measured and reported to the *Monitoring_Manager* where the *Decision_Engine* uses it for deciding on deployments.

4.5.1.3.7 Service Execution

The Service Execution is responsible for creating local instances of services upon request from the *Decision Engine*. The *Service_Execution* is implemented as a python class. An object of this class is created by the SEG. The *Service_Execution* is controlled by the threading where it creates a face for Interest request (push) from *Decision_Engine*. An example of the usage of the class is presented as follows:

`Service_Execution(thread-ID, "thread-name", "seg_ID", namePrefix)`

thread-ID (numeric number, e.g., 1) and *thread-name* (e.g., "Thread-SE") arguments are used to control the threading operation such as terminate or restart process. *seg_ID* is unique ID of each SEG (i.e., SEG_1). The *namePrefix* parameter is used to publish its data content mapping with the NDN face. An example of *namePrefix* can be expressed as */umobile/service_deployment_push/*

The following are the main methods implemented in *ServiceExecution.*

- `face.registerPrefix(namePrefix, onInterest, onRegisterFailed)`: Create a NDN face with given *namePrefix. onInterest* is a response method when *Service_Execution* receives an Interest message matched with a given *namePrefix.*

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

onRegisterFailed is a response function that is executed when *Service_Execution* fails to create a NDN face, it returns an exception error message on the terminal.

- `onInterest(namePrefix, interest, face, interestFilterId, filter):` Process push Interest message (i.e., */umobile/service_deployment_push/*) sending by *Decision_Engine*. The method sends a json file back to the *Monitoring_Manager* along the reverse path forwarding.
- `face.expressInterest(interest, _onData, _onTimeout):` Send pull Interest message to fetch service image from nearby cache or service repository. The name prefix is assigned as *"/umobile/service_deployment_pull/image_fileName" image_filename* is a service name sent by *Decision_Engine.* This name is extracted from the push Interest message inside *onInterest* method. *_onData* is a response method when nearby cache or service repository replies with Data message
- `_onData(interest, data):` Process Data message sent by nearby cache or service repository. This method extracts multiple chunks of Data messages and composes the service image file as tar format.

To deploy the service in the SEG, we develop the *dockerctl* module which communicates with local Docker engine of the underlying hardware. In addition, *temopi* class also uses this module to measure the status of the running containers. The following are the main methods implemented in *dockerctl.*

- `deployContainer(image_fileName):` Deploy new service container. `image_fileName` is a name of the service (e.g. , umobile-store-nano-rpi.tar).
- `get_container_info( ):` Measure status of running containers. This method returns a dictionary, called pi_status containing current status of running containers (see example in Figure 30).

To perform its task, the Service Execution receives image of the service (shown as *Docker image* in the Figure 28 from Decision Engine through NDN face registered with name prefix (i.e., *"/umobile/service_deployment_pull/image_fileName"*). In addition to the image, the Service Service Execution also receives information about how to deploy the image. As shown in the figure, we format the deployment description as a json object (shown as *json object with the deployment description*). The current implementation supports the instantiation of web servers only. The following json object is the skeleton of a deployment descriptor to instantiate service.

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

```
serviceInfo = {
    'umobile-store-nano-rpi.tar':
                    {'image_name':          'al1309/umobile-store-nano-rpi:latest',
                    'port_host': 8081,
                     'port_container': 80,
                     'component': ['busybox.tar', 'python.tar', 'java.tar']},
     'kebapp.tar':
                    {'image_name': 'kebapp:latest',
                    'port_host': 'none',
                    'port_container':'none',
                    'network':'host',
                    'component': ['java.tar']
                    }
```

Figure 31. An example of service description to instruct the Service Execution to instantiate a service

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

4.5.2. INRPP: In-Network Resource Pooling Protocol

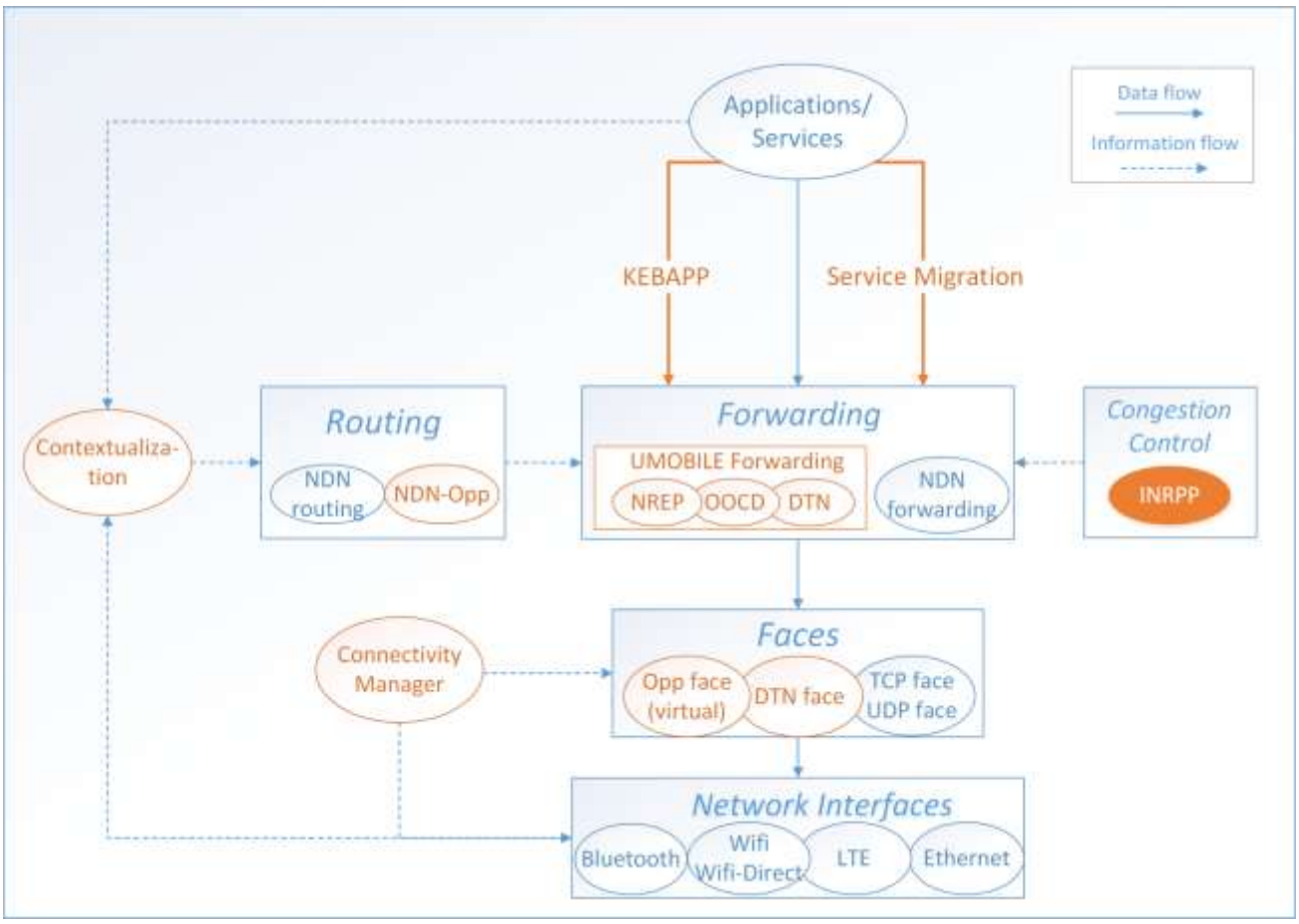

Figure 32. INRPP modules

Within the UMOBILE project, users may need to access Internet services and/or services may need to be pushed to the edge of the network. In this case, it is necessary to deal with congestion to provide services and/or push services as fast as possible. Current end-to-end (e2e) congestion control (e.g. TCP) deals with uncertainty using the "one-out one-in" principle. E2e approaches effectively deals with uncertainty by (proactively) suppressing demand and end-points have to speculate on the resources available along the e2e path.

First NDN/ICN congestion control proposals are based on hop-by-hop congestion control. Despite are based on interest shaping, they still follow the "one-out one-in" principle as well and they do not explore in-network caching capabilities to improve congestion control neither multipath capabilities.

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

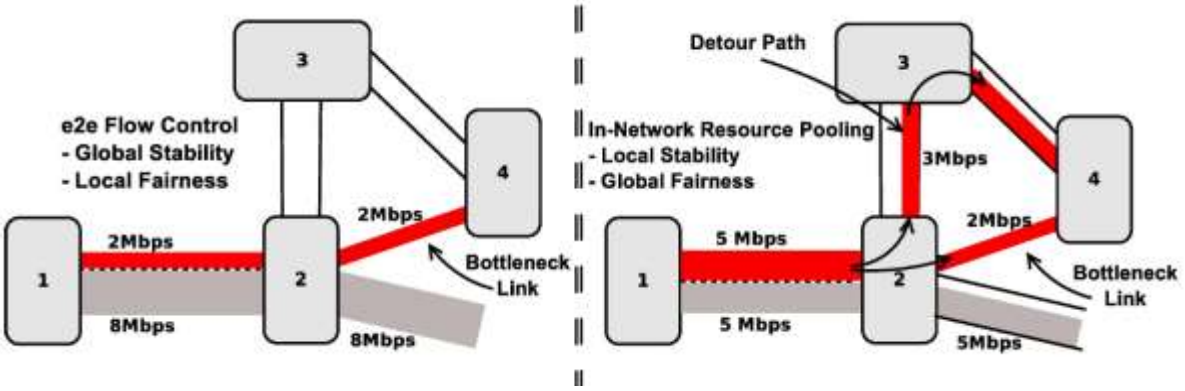

Figure 33. Left: E2E flow control. Bandwidth is split according to the slowest link on the path. Right: In-Network Resource Pooling. Bandwidth is split equaly up to the bottleneck link (global fairness). Detour applies to guarantee local stability

Within the UMOBILE project, we aim to design and evaluate the In-Network Resource Pooling Protocol (INRPP)[15], which pools bandwidth and in-network cache resources in a novel congestion control framework to reach global fairness and local stability, taking profit of the hop-by-hop design and the caching capabilities inherent in the NDN networks, or adding caches (i.e., temporary storage) and breaking the end-to-end principle.

Given this functionality of in-network storage, INRPP comprises three different modes of operation:

1. push: content is pushed as far in the path as possible in an open-loop, processor sharing manner, based on the path's hop-by-hop bandwidth resources to take advantage of under-utilised links.
2. store and detour: when pushed data reaches the bottleneck link, the excess data is cached and simultaneously forwarded through detour paths towards the destination.
3. backpressure: if detour paths do not exist or have insufficient bandwidth, the system enters a backpressure mode of operation to avoid overflowing of the cache. During the backpressure mode, the nodes enter a closed-loop mode, where an upstream node sends one data packet per one received ACK to the backpressuring downstream node.

The INRPP congestion control is out of the scope of WP3 and it is being developed within the Task 4.1 “Providing different levels of QoS and flow controlr of the WP4.

4.5.2.1 Modules to be modified

- Routing module: The routing module should has been modified in order to provide routing information not only for the shortest path, but also for the one-hop detour path to the content.
- Forwarding strategy: The NDN forwarding strategy has been modified in order to forward content using the three modes of operation defined by INRPP.
- Faces: Regular NDN faces have been updated to keep the status of the face (CLOSED_LOOP or OPEN_LOOP) related with the INRPP mechanism.

INRPP protocol has been thoroughly detailed and explained in D4.1 and D4.2. Please refer to these deliverables to read the details of the TCP/IP and NDN implementation of INRPP.

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

The code to simulate INRPP in the ndnSIM [16] simulator is provided in the following repository:

https://github.com/umobileproject/ndnSIM-inrpp

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

4.6. NDN-Opp: Support for NDN operation in Opportunistic Networks

This section provides a description of the support for NDN operation in Opportunistic Networks being developed for the UMOBILE mobile node. Since the NDN framework was selected as the de-facto reference of an information centric networking architecture in the UMOBILE project, the UMOBILE routing engine is embedded in an enhanced version of the NDN framework for Android devices.

NDN-Opp enables communications between devices not relying on any type of infrastructure at all where users can directly exchange content with one another. NDN-Opp offers the capability for devices to exchange data directly with one another in a way that spans beyond the content available from the local neighbourhood. Furthermore, existing D2D communication technologies such as Wi-Fi Direct are based on a connection scheme which requires devices to setup a connection together before exchanging data. Because of the high mobility patterns of devices, NDN-Opp will seek to utilize connection-less transfer of packets given that the establishment of a connection might take too long for the purposes of any meaningful transfer to be performed.

In order to enable other types of applications, we include other functionalities into NDN-Opp that are missing from NDN. For example, Oi! is a messaging app which would benefit from a push-communication mechanism to send messages to specific users. Another variant of push-communication requirement comes from contacting emergency services whereby it is crucial to get the call for help directly to the services. In NDN-Opp, we base the first type of push-model on Long-Lived Interests and the second on Pushed-Data. Both these models are described later in this section. In the context of opportunistic networks, we cannot waste time by having a back and forth exchange before the call for help is delivered.

In this section we describe the *NDN framework for Opportunistic Networks* (NDN-Opp), which is being developed aiming to support opportunistic forwarding based on users' interests and their dynamic social behaviour. The NDN-Opp framework includes some changes in relation to NDN in order to enable social-based information-centric routing over dynamic wireless networks.

The NDN-Opp framework is being developed to fulfil the following UMOBILE requirements, as listed in deliverable D2.2:

R-2: UMOBILE systems MUST be able to exchange data by exploiting every communication opportunity through WiFi (structured, direct), 3G and bluetooth, among UMOBILE systems, operating even in situations with intermittent Internet connectivity.

R-3: UMOBILE systems MUST be able to exchange data taking into account user data interests and context.

R-5: UMOBILE systems MUST have an interface to support the following T3.2 applications: Chat; File exchange/synchronization; Content request/publish.

R-7: UMOBILE systems MUST be able to make messages available to different receivers simultaneously.

R-11: UMOBILE systems MUST provide users only with relevant information, T3.3 i.e., matching user interest.

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

R-15: UMOBILE system SHOULD be able to provide users only with information that matches their interests (e.g. art exhibitions).

R-17: UMOBILE mobile systems SHOULD be able to sense user context (geo- T4.2 location, relative location, proximity, social interaction, activity/movement, roaming, talking) in a non-intrusive manner.

The current specification of NDN-Opp aims to handle the dynamics of opportunistic wireless networks by forwarding interest packets towards best neighbours, which are selected based on their probability to deliver packets to a node carrying the interested data. This information is made available by the Contextual Manager which enables NDN-Opp to know which names the devices in the vicinity are capable of serving. The current specification of NDN-Opp makes use of:

- The existing best route forwarding strategy to deliver interest packets (both normal and long-lived) using a cost based on social weights.
- The standard "breadcrumb" approach to deliver normal data packets based on the information stored on the PIT.
- Best route strategy for forwarding Pushed-Data

The cost of each name prefix (stored in the FIB) is the value of the social weight of the best next hop based on the information provided by the Contextual Manager.

NDN-Opp provides support to the NDN natural pull communication model (e.g. data sharing applications) and to the push communication model used by interactive applications:

- Support for pull model: NDN-Opp allows a receiver to express its interest in receiving some type of data, by sending Interest packets to name prefixes such as /app/Now@/Topic used by the Now@ application (data sharing application being developed to demonstrate NDN-Opp support to pull communication models).
- Support for push communications based on Long-Lived Interest: NDN-Opp allows a node to express its interest in receiving data for a specific application (name prefix) for a certain amount of time. For instance, if a node Nx would like to start using the short messaging application, Oi!, it send a special Interest packet with name prefix /app/oi!/Nx/ or of type /app/oi!/Nx/Ny if node Nx would like to receive Oi! short messages only from node Ny. Unlike a normal Interest which is removed from the PIT either when its Lifetime expires or matching Data is received, this special type of Interest remains in place in the PIT for the duration of its lifetime allowing one Interest to serve more then one data packet.
- Support to push model through Pushed-Data: NDN-Opp allows a node to push Data it is carrying whenever it discovers a new opportunity to do so. A special type of Data packet, called a Push-Data packet can be sent by a node to another one. Such a packet can then be held onto and passed on along a statically configured namespace (e.g. /emergency) without the need to have an Interest in the PIT for it.

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

Unlike NDN, NDN-Opp does not consider that the unavailability of a communication link constitutes an anomaly which must be handled by attempting to seek an alternate way to retrieve the data. In opportunistic scenarios, this wrongfully considers as a problem a natural property of the network environment. In order to address this issue, we introduce the Opportunistic Face (Figure 34) which enables support for intermittent connectivity through the use of a queuing system. Unlike the standard Faces available in NDN, the OppFace stores packets to be sent until the corresponding device comes within transmission range so that they can effectively be transferred over the Wi-Fi Direct link. With this, NDN-Opp has an increased chance of retrieving content that may not be currently in any device within transmission but will become available at some point in the future. An Opportunistic Face is identified by a UUID of the device it corresponds to. There is thus a one-to-one correspondence between an OppFace and some device. If some packets are pending within an OppFace and that corresponding device comes within transmission range, NDN-Opp will pass on the packets that were intended for it.

- An Interest Queue (IQ), which stores Interest packets to be sent.
- A Data Queue (DQ), which stores references to the Data (from the Content Store) to be sent.

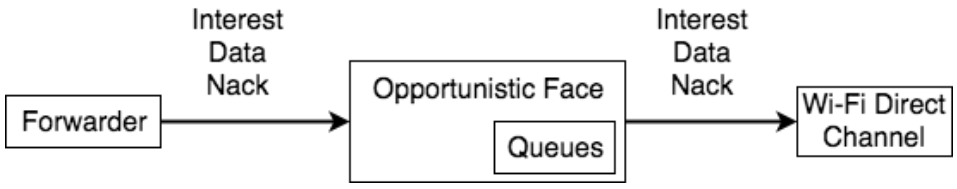

Figure 34. Opportunistic Face

The Opportunistic Face Manager creates Opportunistic Faces after the first contact with a new neighbour node. Furthermore, in order to leverage any opportunity arising from the mobility of devices, the Opportunistic Faces can be used in conjunction with the information provided by the Contextual Manager to pass on pending Interest to newly discovered nodes. When a device comes within range, it advertises the names it can serve data for. Another device detecting this, having some Interests currently pending for some of those names may forward them to that device to improve the chances of obtaining the data (the new device could be better than previously known ones).

One additional functionality of NDN-Opp lies in the support for push-communication models. Within NDN-Opp, we consider long-lived interests and pushed-data to enable two such models. Long-Lived Interest function by maintaining Interests in the PIT even after matching Data has been received in response so that the Interest/Data symmetry is broken and multiple pieces of Data can be received in response to a single Interest. The Pushed-Data functionality allows special Pushed-Data packets to be forwarded without the need to having received a prior Interest for them. This last type of mechanism is crucial to certain types of communication scenarios such as emergency service, where it is unreasonable to require the reception of a prior Interest to enable sending Data.

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

In summary, NDN-Opp encompasses the following novel contributions to the NDN framework in order to allow its operation over opportunistic networks:

- Support for opportunistic communications based on the names advertised by devices.
- Support for push communication model based on Long Lived Interest
    - Users manifest their interest in getting data from some type of application (e.g. short messaging) or from some specific node.
    - Interests persist in the network for as long as established by the Interest source by setting a Long Lived Interest parameter.
    - While Interests persist in the network, data can flow towards the established intermittent path.
- Support for push communication model based on Pushed-Data
    - No need to send an Interest but only to use static information
    - Pushed-Data rely on this configuration to be forwarded along as opportunities arise
- Support for intermittent connectivity of links to other UMOBILE nodes
    - Based on opportunistic faces that include queues of interests and data. The state of such a face (ON/OFF) reflects the presence of the corresponding device within transmission range
- Makes use of:
    - Best route forwarding strategy to handle Interest packets
    - The default “breadcrumb” approach to forward data.
- New fields for the basic types of packets
    - Long-Lived (boolean) for Interest packets
    - Pushed (boolean) for Data packets
    - Modifications to the forwarder for handling those types of packets
- New Wi-Fi Direct channels
- Novel social-aware routing engine which is able to implement different algorithms able to select best routes based on the social weights computed from a social proximity module.

The operation of an NDN-Opp node, as illustrated in Fig. 35, goes over a main set of operations when it gets in contact with another mobile NDN-Opp node (e.g. when a opportunistic face goes

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

UP). As shown Fig. 35 the operation of an NDN-Opp node is similar to the basic NDN operation: the node state is changed by the arrival of an Interest or a Data packet.

The major constraint of the NDN operation over opportunistic networks is the intermittent nature of wireless connectivity. In order to accommodate this property of wireless links, opportunistic faces are used. In this case, each mobile node has a opportunistic face towards to each encountered node (e.g. Face A and C in the case of Node B in Fig 35). Each opportunistic face implements an Interest Queue (IQ) and a Data Queue (DQ).

One may picture Node A as moving within communication range of Node B and passing on its pending Interest as a result. The standard forwarding process at Node B will then determine that Node C is the best choice to retrieve the content that is being requested. Note that this process may also occur dynamically; if Node B, while carrying the Interest encounters a new Node D which, through the information provided by the Contextual Manager turns out to be a good choice to obtain the content, Node B can decide to pass on the matching Interests to it.

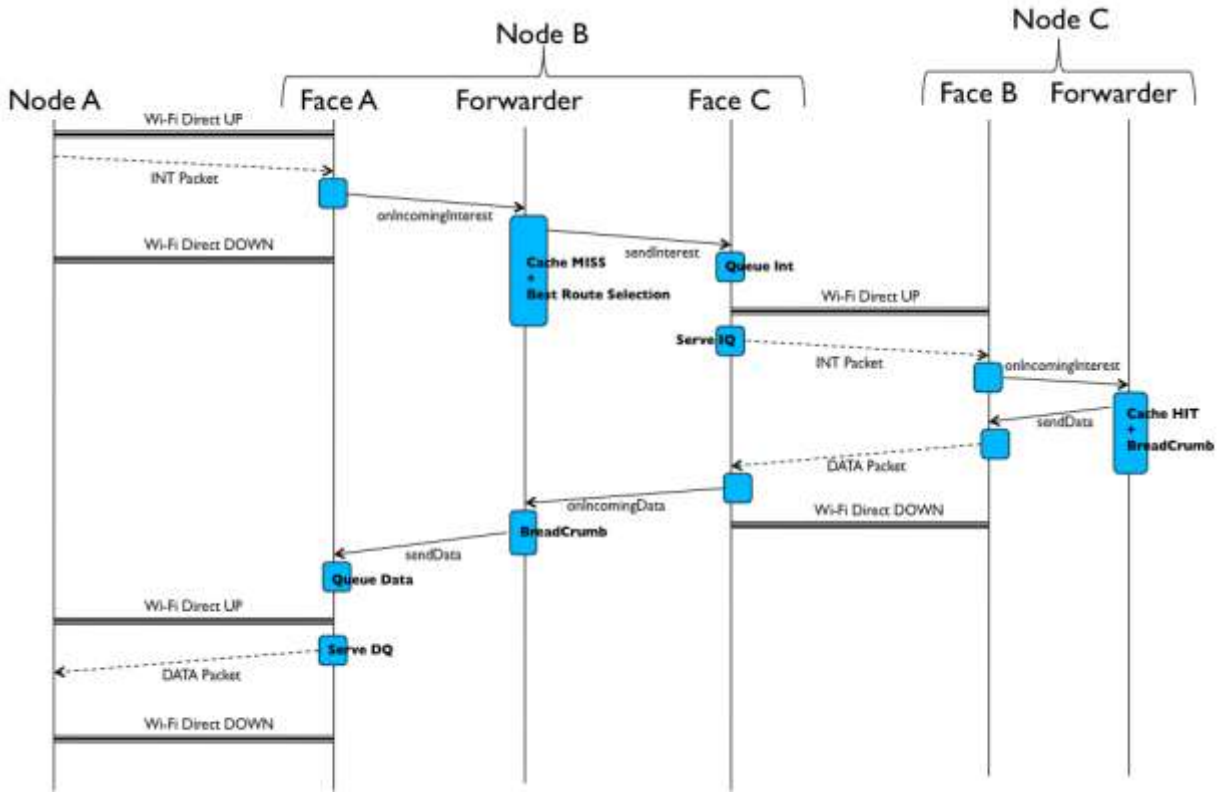

Figure 35. Illustration of NDN-Opp operation

The **forwarder** is able to send and receive packet from several opportunistic faces, as well as the application face. Whenever an Interest packet is received, the standard forwarding process of NDN takes place and results in the Interest being stored in the PIT. At this stage, we rely on the best route forwarding strategy, which performs a normal lookup in the FIB in order to determine which faces the Interest should be sent to.

The **social-aware routing engine** is responsible of the ranking of encountered nodes in their ability to provide content matching names. For this purpose, the routing engine can run any type of social-aware opportunistic routing algorithm [2,12].

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

Computing routing information in opportunistic network based on social interactions has great potential as less volatile proximity graphs are created [13].

For example, the TECD[10] algorithm aims to compute the Social Weight (SW) of a node towards a name prefix (which can represent another node, such as used by short messaging applications). By computing SWs, the TECD algorithm is able of capturing the evolution of social interactions in the same daily period of time over consecutive days, by computing social strength based on the average duration of contacts. The TECD algorithm can also be used to capture the Importance of any node (TECDi) in a daily sample, based on its social strength towards each user that belongs to its neighbour set in that time interval, in addition to the importance of such neighbours. The TECDi is based on the PeopleRank function [14]. However, TECDi considers the social strength between a user and its neighbours encountered within a daily sample, while PeopleRank computes the importance considering all neighbours of that node at any time.

NDN introduces more freedom in the forwarding plane. As a consequence, the routing can be less restrictive, leave loop detection to forwarding time and not worry about convergence time. Many solutions have been proposed for forwarding schemes in opportunistic networks in the past. However, while mobility has an impact on the performance of routing in such scenarios, the stability offered by considering social metrics seems to offer potential to include routing scheme into NDN for those types of networks. Regarding the mechanism underlying the routing engine, prior work has been done on the problem in the context of NDN but the applicability of existing proposals is unclear at this stage. Those proposals are all typically based on existing algorithms for host-based networks (e.g. OSPF, OLSR, Hyperbolic) where the IP prefixes are merely replaced by Name prefixes. In the context of the UMOBILE project, one line of investigation is based on NLSR with the addition of a filtering mechanism for the advertised attachment of prefixes associated with a measure of reachability of the data (K) through it. That cost is updated upon advertisement. The idea is that this filtering could make it scalable. This would enable devices to disseminate prefixes locally (i.e. upon encounter) and limit the scope of their propagation. The forwarding decisions are thus based on a combination of K with contextual information; neighbour degree centrality (A), neighbour availability (U) and time lapse between sending an Interest and receiving Data (D). The first two are provided by the Contextual Manager while the third is a measure performed by the forwarder. These three indicators, combined with the cost K will drive the forwarding of Interest packets. The downward path criterion is used to reduce the likelihood of routing loops. This is still a preliminary line of investigation. Others are explored.

The motivation to introduce a routing scheme is aligned with the NDN forwarding approach which expands the intelligence of devices by relying on strategies. The existing forwarding pipelines of NDN are left untouched, except the strategy which might be applied to Pushed-Data.

The routing will enable to transfer packets between any types of faces, whether they correspond to other UMOBILE nodes or hotspots. This means that an Interest packet will be able to originate in a device opportunistically. When another device gets that Interest over opportunistic link, it can run into a hotspot at a later point, detect that it is an interesting way to get it to a data source and send it over the Wi-Fi link. Once the infrastructure network returns the data to the hotspot, it will be carried back to its original requester which can rely on NDN-Opp to do so. In order to improve this process on the Data path, this is where the breadcrumb might be abandoned given

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

that the requirement for the original device who passed the Interest to the hotspot to collect the Data might be too strong.

The social-aware routing engine is responsible for the computation of social weights based on information collected about neighbour devices. For that the routing engine makes use of an interface towards the Contextual Manager developed by the UMOBILE architecture.

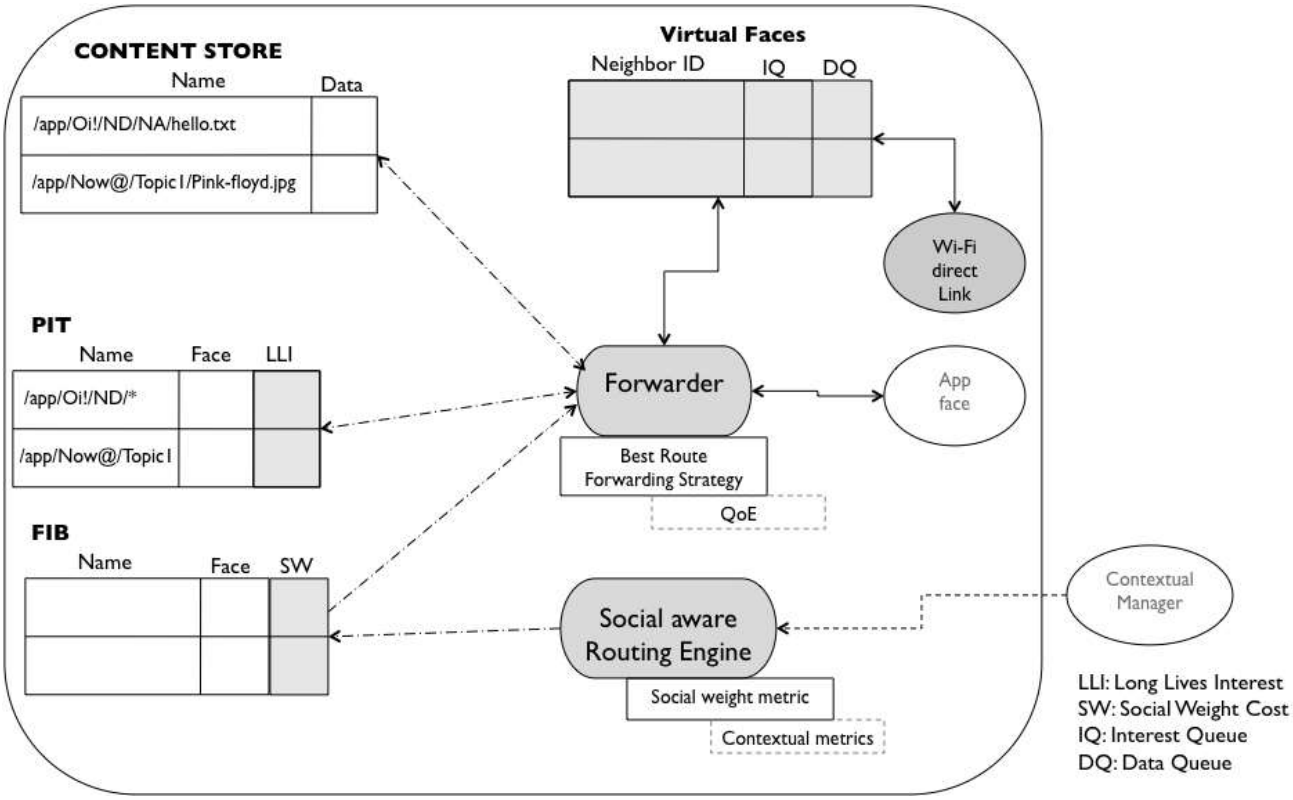

Figure 36. NDN-Opp design

The **Contextual Manager** shall be able to provide information about neighbour devices, to ensure the proper operation of the routing engine. The basic operation of the NDN-Opp routing engine requires information about the average contact duration between pair of nodes in specific daily samples, as illustrated in Fig. 37.

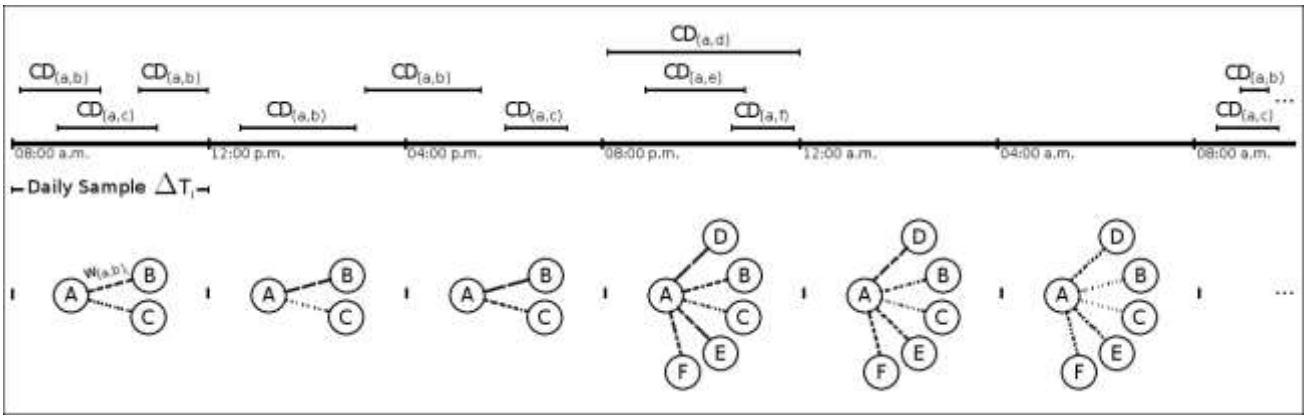

Figure 37. Contacts a node A has with a set of nodes x in different daily samples

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

The operation of the **social-aware routing engine** is based on a social weight measurer, a social weight repository and a gatherer of information about social weight and data carried by neighbour nodes:

- **Social Weight Measurer (SWM)** – responsible for keeping track of the contact duration information provided by the Contextual Manager. SWM maps this duration to the name prefixes carried by the encountered nodes. Based on that, this component computes the level of social interaction of the current node towards the name prefixes carried by encountered peers. This social weight determines how good the encountered node is to reach nodes that carry data related to such name prefixes. The social weight computation takes place at every hour as to allow for a better view of the dynamic social behaviour of users.
- **Social Weight Repository (SWR)** – responsible for storing the list of name prefixes that the current node comes across (obtained upon encountering a peer). SWR holds the name prefix, time it has been first encountered, and the cumulative duration of within a specific hour. In the case, the current node still 'sees' such name prefix (i.e., respective peer node is still in the vicinity), the 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 name prefixes and social weight towards them of encountered node. This element is also responsible for obtaining information concerning the content carried by encountered nodes.

The computation of social weights is triggered when a specific face notifies the SWM of the presence of a neighbour node. The SMW creates entries for this encountered node in the SWR considering the name prefix information obtained by the SWDCG. The SWM keeps track of the contact duration between current and encountered nodes by querying the Contextual Manager and updates the SWR accordingly. Since nodes (i.e., their users) present different patterns of behaviour in different time periods, the computation of social weights takes place in an hourly fashion. Information obtained by the SWCDG from the encountered node is used to populate PIT and the content store.

Future work in the development of the NDN-Opp framework will be done based on exploiting different contextual information provided by the Contextual Manager for the development of novel routing approaches. The major action points are:

- **Interface with the Contextual Manager**: The collection of context information about neighbour nodes, done currently by scanning the Bluetooth face, should be done also by scanning the WiFi direct face. The goal is to allow the collection of neighbour information to be done over the same wireless range as the one used to exchange data. Subsequent evaluation about the performance of both approaches will be done based on a UMOBILE testbed.

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

Novel opportunistic routing mechanism. In order to improve the quality of the routes used to forward Interests, we will investigate an alternative proposal for the NDN-Opp framework based on novel strategies to forwarding interest packets in dynamic networks. In a first stage we will keep using the NDN default "breadcrumb" strategy to forward data. In a second stage we will investigate a more intelligent method also to forward data, aiming to exploit any contact with a neighbour node able of forwarding data towards the receiver(s), and not just the wireless contact with the node that previously sent the Interest packet (which in the approach followed in the default NDN "breadcrumb" approach). The performance of both alternatives will be assessed in real scenarios.

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

5. UMOBILE applications

5.1. PerSense Mobile Light (PML)

PerSense Mobile Light (PML)[7] is an application that has been developed to assist researchers in analysing network context. PML is an example of an external application that the Contextual Manager can be plugged to, to collect data. The purpose is to show how the Contextual Manager can be easily extended to collect new parameters, by relying on external applications.

PML has been released in Android on May 2015, and addressed in deliverables D3.1; D3.3; D5.3; D5.1. It is currently being used in experiments to collect roaming data. Traces collected are expected to be provided openly in the context of UMOBILE. PML is also available to researchers worldwide in the UMOBILE Lab (as one of the applications that can be remotely used).

In its current format (v3.0), PML has been changed to capture information concerning peers via Wi-Fi Direct as well as via Bluetooth. Furthermore, PML generates automatically (csv format) daily roaming reports, which a user can then obtain via e-mail or simply downloading such reports from the device.

Usage guidelines

PML is available for Android devices via the Google AppStore. A user simply needs to install it in its device. The current PML 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 WiFi Direct as well as via Bluetooth). This data is kept locally only on the user device, and for 7 days. Every day, three csv reports are generated[8]:

- **Visited networks' report**. Each entry consists of the tuple <id, SSID, BSSID, timeon, timeout, dayoftheweek, hour>
  - Id: sequential waypoint identifier.
  - SSID: Service Set Identifier.
  - BSSID: Basic Service Set Identifier.
  - Timeon: timestamp (milliseconds) when the device first connected to the AP.
  - Timeout (milliseconds) when the device lost attachment to the AP.

7 https://play.google.com/store/apps/details?id=com.senception.persenselight&hl=en

8 In version 3.0, the reports are still generated without obfuscation. In version 4.0, MACs and SSIDs are obfuscated; strings identifying devices also.

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

- Dayoftheweek: represented by an integer, where 1 corresponds to Sunday, and 7 to Saturday.
- Hour: hourly slot (24 slots).

- **Peer report.** Each entry consists of the tuple *<id, DeviceName, MAC, dateTime, lat, long>*
  - Id: sequential waypoint identifier.
  - DeviceName: Device identifier, string.
  - MAC: Wi-Fi MAC address of the device.
  - dateTime, string representing the day and instant when the device first entered the range of an AP (DD-MM-YYYY at HH:MM:SS)
  - lat, long: latitude and longitude of the location where the current device encountered the peer device.
- **Waypoints report**. Each entry consists of a tuple with the format: *<id, bssid, dayoftheweek, ssid, attractiveness, dateTime, latitude, longitude>*.
  - Id: sequential waypoint identifier.
  - Dayoftheweek: string, Mon to Sun.
  - ssid: Service Set Identifier
  - Attractiveness: ranking of the AP, derived from contextualization[9].
  - dateTime, string representing the day and instant when the device first entered the range of an AP (DD-MM-YYYY at HH:MM:SS)
  - lat, long: latitude and longitude of the AP.

5.2. Short Messaging Application

The Short messaging application Oi! was ported to operate on top of NDN-Opp (described in section 4.6). The Oi! application allows users to send messages to each other in an opportunistic manner using push-communications over NDN (by using the capabilities included in NDN-Opp). Through the UI Oi! interacts with NDN-Opp, which is responsible for managing all the messages

In the current version, attractiveness is set to 1 if the device connects to an AP, and 0 if it simply crosses an AP. In v4.0, attractiveness is set via an automatic ranking function derived from learning of habits, usage of the AP, and QoS parameters as well as availability of devices.

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

to be sent, as well as received messages. The primitives `Send(destination, message)`, allows the application to pass to NDN-Opp the data needed for this one to build a message to be sent to a given destination node; *Receive(message),* allows the application to get from NDN-Opp a received message; and `GetContactsList()`, allows to fetches a list of known contacts to display on the UI. Also, Oi! will also support `received` and `read` notifications which enables acknowledging messages between Oi! users.

Fig 38 shows how Oi! works on top of NDN-Opp, to allow users to opportunistically exchange messages based on their level of social engagement. So, the first check (1) is whether NDN-Opp is available. If not, the application asks the user to proceed with its installation (2). Otherwise, Oi! connects to it.

Once connected to NDN-Opp, Oi! displays 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 NDN-Opp (8).

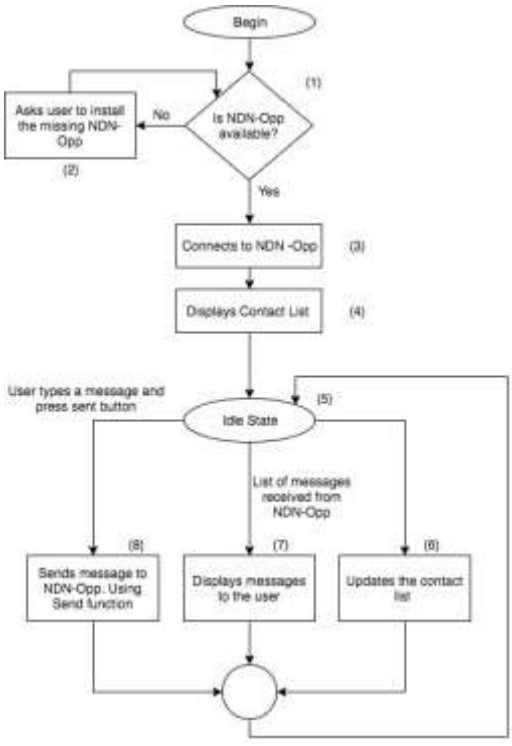

Figure 38. Oi! application operation flowchart

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

For a full description of Oi! code documentation, refer to Annex 1. The open source code of Oi! is available at GitHub: https://github.com/COPELABS-SITI/Oi

5.3. Content Sharing Application

The content sharing application Now@ was developed to work on top of NDN-Opp (described in section 4.6). Now@ is an open-source application developed for Android that allows users to exchange information such as text, images and documents over an NDN infrastructure, using predefined categories that help users define the interests they want to communicate about. To do this, we use ChronoSync to carry out the transfer and synchronization data between each of the users that uses the application even in condition without internet connectivity. The primitives `Send(prefix, message)`, allows the application to pass to NDN-Opp the data needed for this one to build a content to be sent to the network; *Receive(content),* allows the application to get from NDN-Opp a received content to display on the UI.

Fig 39 shows how Now@! works on top of NDN-Opp, to allow users to opportunistically exchange content based on their interests. So, the first check (1) is whether NDN-Opp is available. If not, the application asks the user to proceed with its installation (2). Otherwise, Now@! connects to it.

Once connected to NDN-Opp(3), Now@ starts ChronoSync(4) and displays categories which users can select (5) and shifts to idles state (6), in which one of the following actions may take place (order is irrelevant and serves for the purpose of explanation only):

Content is received: Now@! displays the incoming content to user (7).

User creates a content: Now@! forwards the newly composed content to NDN-Opp (8)

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

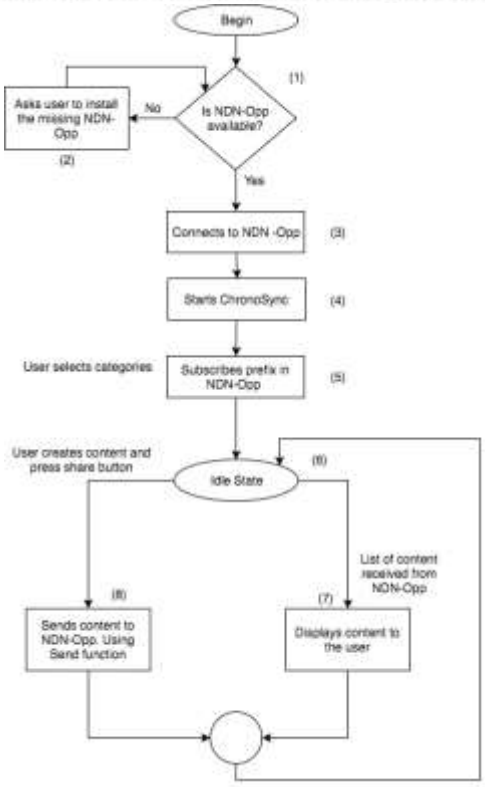

Figure 39. Now@! application operation flowchart

The open source code of Now@! is available at GitHub: https://github.com/COPELABS-SITI/NowAt

6. Conclusion

This document covers the progress of the project so far towards 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 we have developed, their status and the next steps for their integration. For all components that we have deployed, we also attach the relevant code, as well as their manuals.

All individual modules are the building blocks that lead us towards reaching our final goal of a unified information-centric, delay-tolerant networking platform offering extended services to its users. Further optimizations will be performed during the integration, optimization and evaluation period.

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

References

[1] Waldir Moreira, Manuel de Souza, Paulo Mendes, Susana Sargento, "Study on the Effect of Network Dynamics on Opportunistic Routing", in Proc. of AdhocNow, Belgrade, Serbia, July 2012

[2] Waldir Moreira, Paulo Mendes, Susana Sargento, "Social-aware Opportunistic Routing Protocol based on User's Interactions and Interests", in Proc. of AdhocNets, Barcelona, Spain, October 2013.

[3] Waldir Moreira, Paulo Mendes, Susana Sargento, "Opportunistic Routing based on daily routines", in Proc. of IEEE WoWMoM workshop on autonomic and opportunistic communications, San Francisco, USA, June, 2012.

[4] Ioannis Psaras, Lorenzo Saino, Mayutan Arumaithurai, KK Ramakrishnan, and George Pavlou. Name-based replication priorities in disaster cases. In Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on, pages 434{439. IEEE, 2014.

[5] George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos, A Survey of Information-Centric Networking Research. Communications Surveys Tutorials, IEEE, 16(2):, Second 2014.

[6] J Seedorf, M. Arumaithurai, K. K. Ramakrishnan, and N. Blefari Melazzi, “Using icn in disaster scenarios,” IRTF, 2015. [Online]. Available: https://datatracker.ietf.org/doc/draft-seedorficn-disaster/

[7] D. Kutscher, S. Eum, K. Pentikousis, I. Psaras, D. Corujo, D. Saucez, T. Schmidt, and M. Waehlisch, “Icn research challenges,” IRTF, 2014, August 2014.

[8] O. Ascigil, V. Sourlas, I. Psaras and G. Pavlou, "Opportunistic off-path content discovery in information-centric networks," *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, Rome, 2016, pp. 1-7. doi: 10.1109/LANMAN.2016.7548860

[9] I. Psaras, S. Reñé, K. V. Katsaros, V. Sourlas, G. Pavlou, N. Bezirgiannidis, S. Diamantopoulos, I. Komnios, and V. Tsaoussidis. 2016. Keyword-based mobile application sharing. In *Proceedings of the Workshop on Mobility in the Evolving Internet Architecture* (MobiArch '16). ACM, New York, NY, USA, 1-6. DOI: http://dx.doi.org/10.1145/2980137.2980141

[10] Waldir Moreira, Manuel de Souza, Paulo Mendes, Susana Sargento, "Study on the Effect of Network Dynamics on Opportunistic Routing", in Proc. of AdhocNow, Belgrade, Serbia, July 2012

[11] Waldir Moreira, Paulo Mendes, Susana Sargento, "Social-aware Opportunistic Routing Protocol based on User's Interactions and Interests", in Proc. of AdhocNets, Barcelona, Spain, October 2013

[12] Waldir Moreira, Paulo Mendes, "Social-aware Opportunistic Routing: The new trend", Springer Book on Routing in Opportunistic Networks, ISBN 978-1-4614-3513-6, August 2013

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

[13] Waldir Moreira, Paulo Mendes, “Dynamics of Social-aware Pervasive Networks” in Proc. of IEEE PERCOM workshop (PerMoby), St. Louis, USA, March 2015

[14] A. Mtibaa, M. May, M. Ammar, and C. Diot, Peoplerank, “Combining social and contact information for opportunistic forwarding” ,in Proceedings of INFOCOM, (San Diego, USA), March, 2010.

[15] I. Psaras, L. Saino, and G. Pavlou. "Revisiting resource pooling: The case for in-network resource sharing." Proceedings of the 13th ACM Workshop on Hot Topics in Networks. ACM, 2014.

[16] S. Mastorakis, A. Afanasyev, I. Moiseenko, L. Zhang, "ndnSIM 2.0: A new version of the NDN simulator for NS-3", NDN Technical Report NDN-0028, 2015.

[17] Android port of Named Data Networking Forwarding Daemon https://github.com/named-data-mobile/NFD-android

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