R-P2P: a Data Centric DTN Middleware with Interconnected Throwboxes

Francesco De Pellegrini, Iacopo Carreras,
Daniele Miorandi and Imrich Chlamtac
CREATE-NET
via Alla Cascata 56/c
38100, Trento, Italy
name.surname@create-net.org

Corrado Moiso
Telecom Italia Lab.
Via G. Reiss Romoli 274
10148, Torino, Italy
corrado.moiso@telecomitalia.it

ABSTRACT

In this paper we describe R-P2P, a novel system meant to support the search and retrieval of data in a certain location. R-P2P couples opportunistic wireless communications between mobile devices with an interconnected network of throwboxes. The set of throwboxes implements a distributed and localized data storage.

Overall, the system integrates in an on-demand, delay-tolerant fashion two main network extensions: a query forwarding engine running on top of the opportunistic network and a data retrieval mechanism performed on throwboxes.

We detail the building blocks of the proposed system, describing the functionalities and the interactions of the various middleware modules. Finally, we illustrate the current implementation of R-P2P.

Keywords

Middleware, Opportunistic Communications, Delay Tolerant Networks, DHT

Categories and Subject Descriptors

C.2 Computer-Communication Networks [**C.2.1 Network Architecture and Design**]: Wireless communication

General Terms

Design

1. INTRODUCTION

The diffusion of electronic devices equipped with computing and wireless communications capabilities has changed the notion of interaction with the technology in everyday life. The evolution of information technologies is not always linear, though. One instructive example is the success of guidance systems, traditionally based on the GPS technology. GPS technology, in fact, was designed intentionally for positioning systems, and requires an extremely expensive satellite infrastructure. But, surprisingly, such technology is facing unexpected competitors. Cellular phones, for example, can provide positioning functionalities based on base stations triangulation, and projecting detected positions on a map. The resolution of this technique is on the order of a kilometer at present, and of course it is not suitable for vehicular road guidance systems. Nevertheless, it is applicable to a range of less demanding location based applications: in general, consensus exists on the fact that the massive deployment of communicating and computing devices has the potential to be exploited in novel and unplanned manners.

This relates to the need of coupling environmental or context data to one or more services running on user devices. In practice, data of interest for an application can be retrieved via available sensor measurements or directly from devices in the surroundings. For a wide class of applications, in particular, a key feature of the available data, e.g. the customary example of a temperature measurement or the position of the closest Starbucks coffee, is the locality of such information with respect to the user position. A self-explanatory example is represented by the fast growing field of Bluetooth advertisement (see for example `http://www.kombok.it` and `http://http://www.bluetooth-advertising.co.uk`). To this aim we use the generic term *source* to mean a wireless device diffusing information pertinent to a given location.

Translated into networking requirements, location based data distribution would require a fine-grained deployment, bridging possibly different technologies in order to reach the needed wireless coverage. But, at present, such a fine-grained wireless coverage has a unique candidate, i.e. a cellular network. The idea of injecting and distributing local data into the operators' networks, poses several technical concerns though, especially in terms of scalability.

Nevertheless, alternative architectures are possible. Recently, several legacy devices, i.e. standard cellular phones, have been enabled with proximity wireless communications. In particular, Bluetooth or IEEE802.11 interfaces would potentially lead to systems composed of several handheld devices coming now and then into radio range, forming a mobile ad hoc network based on one-hop communications. These systems are referred to as opportunist networks. Their con-

This work has been partially funded by the European Commission within the framework of the BIONETS project EU-IST-FET-SAC-FP6-027748, `www.bionets.org`.

AUTONOMICS 2008, September 23-25, Turin, Italy

DOI 10.4108/ICST.AUTONOMICS2008.4598

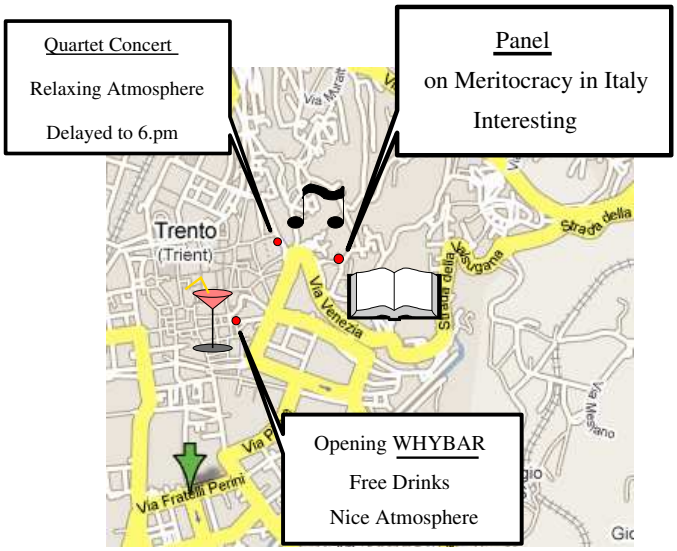


Figure 1: A graphical representation of the use case, enriching traditional mapping systems with notifications based on user-generated contents.

nectivity pattern is typically intermittent due to sudden and repeated changes of environmental conditions: nodes moving out of range, drop of link capacity or short end-node duty cycles [5, 9]. In this paper we consider a scenario where several wireless mobile devices cooperate in order to feed local data to applications leveraging opportunistic communications. The opportunities for pairs of nodes to communicate are rather short and, to this respect, communication among nodes is conveniently described by the set of nodes' *contacts* [4, 10].

The use of such systems appears promising for two main reasons. First, there is no need for central data management. Second, it is inherently localized, because only communications with nodes in radio range are possible. Of course, even though they natively support data diffusion, opportunistic communications have several limitations. In terms of delay, for example, no guarantee can in principle be provided so that applications running on top of such networks are required a certain degree of *delay-tolerance*. Furthermore, the amount of data that can be exchanged per contact is dictated by the finite duration of contacts, the need to trade off device discovery duty cycles for energy savings, and, last but not least, by the limited bandwidth of current technologies, e.g. Bluetooth. As it will be clear in the following, for the above reasons we enforced the opportunistic network with an interconnected network of throwboxes.

The main contribution of this paper is the description of a system, namely R-P2P (Reverse Peer to Peer) and the related middleware interfacing mobile devices and throwboxes. The baseline functionality provided by R-P2P is the retrieval of data generated by sources deployed in the area of interest to the user. R-P2P is novel since it decouples the query diffusion and the data retrieval process using two separate network extensions: an opportunistic network supports the diffusion of queries and advertisements, whereas a network of interconnected throwboxes supports the retrieval of data injected into the system.

In the following description of the architecture, cooperation and security issues are out of the scope of the paper; nevertheless, enforcing users' cooperation and ensuring security are a clear requirement for the R-P2P system.

Use case

As a reference scenario we represent a simple use case. Our virtual user Sonia is visiting Trento, Italy, during the end of May. She came to town in order to attend the Festival of the Economy, a known event involving Italian institutions, the local municipality and several Italian and foreign companies who organize events for the whole duration of the 3-days festival.She knows that, apart from the scheduled institutional events, there are several satellite social events occurring in parallel, some of them with short or almost no-advice before.

This scenario fits the typical requirements for the usage of the novel system described in the following:

- during a limited period, a high density of outside people gather in town; they are typically interested in information relative to the specific location and in events occurring in the area – this requires an *easy to deploy and lightweight architecture*;
- visitors have a broad range of interests, covering events related to the festival, but also satellite events occurring on ad-hoc fashion, e.g. restaurants special offers, wine bar or parties – this requires a *general purpose data sharing* system;
- each user will use local information to decide on the eligibility of some event for his/her schedule, but *most relevant contents will be generated by users on the fly, including opinions, tastes and ratings* – this excludes web-based only publishing solutions;

R-P2P is designed to meet the above requirements. While Sonia walks in Trento, her handheld device running R-P2P will forward queries for certain information, which might become available at some of other person's devices running R-P2P; R-P2P will also allow transparent exchange of data during contacts. Sonia will verify from time to time the dis-

play of her handheld device: an application will integrate fresh information on ongoing events and information gathered by other users on a map, as depicted in Fig 1.

In the example, Sonia will be interested in a concert and a talk: based on some user's feedback, she will opt for the concert. She will also obtain some multimedia material on the place the concert will be hosted in, a nice palace downtown Trento. At the concert, she will confirm the opinion of other visitors, just filling a short rating form on her smartphone. Later on that evening, she also will be able to plan an unexpected stop at wine bar, whose owner is promoting the opening with free drinks; the owner just advertised the happening over the R-P2P network, diffusing some adds and storing some related multimedia material.

Paper structure

The work is organized as follows. In Sec. 2 we revise related works, whereas in Sec. 3 we provide a general description of the system architecture. In Sec. 4 and Sec. 5 we address the functional blocks composing the two main devices of the architecture, i.e., user nodes and throwboxes. In Sec. 6 we describe the initial implementation of the mobile opportunistic network, based on the U-hopper middleware. The last section is devoted to conclusions and future directions.

2. RELATED WORKS

The possibility of running applications on top of opportunistic networks is recently gathering some attention. In [16] the operation of application protocols such as HTTP or email in delay tolerant networks (DTNs) is addressed: the issue there is how to adapt them in order to run under a regime of intermittent connectivity. A dedicated application-aware module pilots storing and forwarding operations of DTN modules using a suitable "application-hint" header protocol. [15] describes general principles for such adaptation.

We notice that the system considered in this paper is not end-to-end, but data centric, so that traditional end-to-end DTN techniques do not apply.

Other authors have explored the technical feasibility of P2P systems in presence of intermittent connectivity. The work in [17] proposes a system, namely $7DS$, which runs as an application over mobile hosts. The aim of $7DS$ is to provide enhanced Internet connectivity sustaining cooperative data exchange with mobile peers and a mechanism for data dissemination. $7DS$ implements a passive/active scheme and mobile host rely on local peer connections to retrieve data from local servers. In [11], authors introduce passive distributed indexing (PDI), a P2P overlay designed for mobile applications. PDI provides a query diffusion protocol on a restricted overlay of mobiles covering a few hops. The work in [22] proposes a delay-tolerant overlay for a mobile ad-hoc network, namely ASOS, with the aim of accomplishing disruption tolerant operations. In ASOS queries and data are disseminated on a MANET: several issues are explored on data diffusion and optimization of data retrieval.

Several related works, in particular, proposed solutions to cope with frequent disruptions using cooperative caching techniques [18, 23, 12, 7]. The main effort is to sustain data flows even in presence of high route failure rates, inherently due to mobility [7]. Authors of [18] proposed an hybrid technique using both peer-to-peer communications and APs in radio range. Also, in [23], authors describe existing tradeoffs depending on the mobility and cache size trading off local caching and queries issued to central-servers. Techniques for probabilistic quorum construction are applied to ad-hoc networks in [12], based on the seminal work [13].

2.1 Novelty of the proposed architecture

The R-P2Psystem works based on diffusion of queries and avertisements and on data retrieval. Most related works mentioned before share the common view that *both* queries diffusion *and* data retrieval occur top of the same MANET. We take a different approach for two main reasons. First, at the current status of the technology, measurements performed on the field show that the duration of contacts is not always such as to guarantee complete data exchange for large data files [8]. Also, a further problem holds with respect to data retrieval: at the time when a query reaches a node able to retrieve the queried data, the potential return path through which the query is forwarded likely disappeared. For such a reason, all the above mentioned approaches typically foresee a large overhead in terms of data dissemination: basically they try to reach back the source of the query via controlled flooding.

In sight of the above technical concerns, the major novelty of the proposed system architecture is that it couples a DHT over an interconnected network to an DTN network of mobiles. The middleware described in the following, in particular, is meant to confine the forwarding of queries and advertisements to the opportunistic part of the network, whereas the data retrieval is operated through a set of interconnected throwboxes. The net effect is that *we split the forward and the return path used in the query/retrieve data process.* Also, the proposed technique leverages the opportunistic part of the network for queries and advertisements, which are typically very short messages, whereas data retrieval is operated via the interconnected network. Another feature of the proposed solution is that the system architecture is designed to retrieve natively data from local sources in radio range, and to maintain the local scope of the retrieved data.

We called this middleware Reverse Peer-to-Peer (R-P2P), because the system resembles a P2P system, but opposite to a regular p2p system, data are injected into the storage system after the query is produced, whereas in regular p2p system, queries match data *already* stored in the system.

3. SKETCH OF THE SYSTEM

The architectural skeleton of the proposed architecture is depicted in Fig. 2. We consider three types of communication devices: *throwboxes*, *user nodes* and *sources*. User nodes are mobile handheld devices, such as a mobile phone, a PDA or a similar electronic device with communication capabilities, carried around by users. Typically, such devices are equipped with a primary wireless interface connecting to an operator network, and a secondary wireless interface, e.g. Bluetooth or IEEE802.11. When in radio range of another device, using these secondary interfaces, user devices can establish communications in a point to point fashion with no need for the fixed infrastructure. This is usually referred to as opportunistic communications (see [10, 9]).

Also, user devices can interact with sources in radio range, namely devices acting as sources of *local* data: these data are consumed by applications running on user devices. Sources are depicted in Fig. 2 at the bottom layer: a typical example of such a device can be a laptop equipped with an RF interface acting as a bucket of user-generated content, but

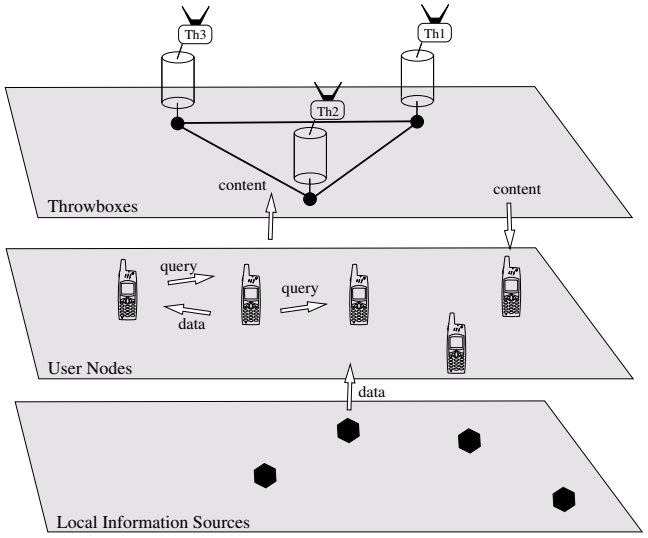


Figure 2: Layered representation of the network components: on the upper layer, throwboxes, in the middle layer user nodes and in the lowest layer localized storage.

we plan to include sensors or RFIDs in the middleware as well[1]. In some other cases, contents might be generated directly onboard of user nodes, e.g. this is the case of a video or a picture taken with a video-phone.

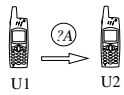
A set of *throwboxes* occupy the third logical layer of the system. Throwboxes can interact within themselves via a conventional interconnected network, e.g. a mesh network. User nodes communicate with a throwbox using the secondary wireless interface.

Notice that the basic architecture of the system described here is similar to what proposed in [19] for data collection with the use of "mules". In that work, though, data collected by "mules" are finally uploaded to the sink of a sensor network; the use of a query-based mechanism makes R-P2P different and novel, because queries are generated at user nodes and the sink here does not correspond to the origin of the sources. Also, R-P2P is meant to support both `query` and `publish` operations. In particular, issuing a `query`, a mobile device can retrieve data pertinent to the given location where the user is located. Conversely a user node can publish information relevant to the given location in the form of an advertisement.

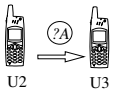
3.1 Query diffusion and data retrieval

In order to implement the two basic `query` and `publish` operations, two main mechanisms have to be implemented: query/ advertisement diffusion and data retrieval. The pictorial representation of the two mechanisms is reported in Fig 3 in the case of `query` operation. The query/advertisement diffusion is carried at the user device layer by leveraging opportunistic forwarding. To this aim several variants are possible, including K-relaying and spray and wait, see [6, 2, 20] and reference therein. As we mentioned before,

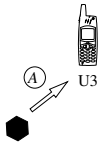
[1]The issue in such cases is to have a suitable wireless card for the user nodes to interface with those devices, which is beyond the scope of the current work.



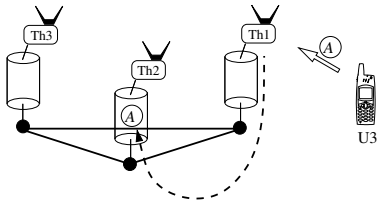
(a) U1 issues a query for data A to node U2



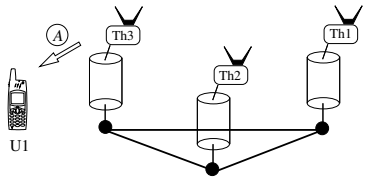
(b) U2 forwards the query to node U3



(c) Data A retrieved by used node U3 from a source



(d) Data A stored to the DHT via Throwbox Th1 by U3



(e) Data A retrieved at Throwbox Th3 by user node U1

Figure 3: Sketch of the system operations. In the example it is reported the set of interaction from the query of a given data to the retrieval at a throwbox some time after the query is issued.

in order to efficiently use the capacity of opportunistic networks, only queries or advertisements should be propagated. In fact, data retrieval is performed only with devices in range and no data are forwarded in a multihopping fashion (notice that one hop exchange of data is possible though). The typical operations for query/advertisement dissemination follow:

- once a new `query` is issued, a unique query identifier `query-id` is generated, and attached to the `query`; the identifier will later on permit to the node issuing the query to recognize the corresponding data once available; also, optional information released by the source node will provide indication about the intended validity of the query and/or the data and the geographical scope, i.e. the maximum distance within which the query and the potentially retrieved data are considered of interest.

- two user devices coming into reciprocal radio range first check whether they possess data which fulfill the other node pending queries, i.e. queries they generated before; if so they exchange the corresponding data and

erase pending queries. As a second step, each user device forwards pending queries and registers new queries received by the peer node (Fig 3(a) and (b));

- when in range of a source, a user device queries the source in order to retrieve data of interest according to stored queries (Fig 3(c)).[2]

Thus, in R-P2Pdata retrieval occurs *on-demand*; retrieved data are thereafter made available over a distributed storage system:

- when in radio range of a throwbox, a user node uploads source data retrieved, and erase them from its local storage (Fig 3(d)) – uploaded contents are associated to the `query-id` that triggered their retrieval;
- at throwboxes layer, a complete overlay permits to share data uploaded into the system;
- after disseminating queries, user devices coming into contact with a throwbox check own pending `query-ids` for matches with the data retrieved by the system and download the corresponding data (Fig 3(e)).

A dual functionality is the diffusion of advertisements: a user node can upload data into a throwbox, and advertise the presence of such data in the throwbox storage system diffusing an advertisement message. Advertisements are tagged with a topic description and an identifier to retrieve (additional) content on the throwboxes. We skip the detailed description for space reasons.

4. FUNCTIONAL BLOCKS

The identified architecture has been in part implemented by a set of functional blocks as depicted in Fig 4. As concerns user nodes, 4 main modules exist:

- Service Container: it executes the service logic, which includes services deployment, execution deprecation and update. Furthermore, this component issues queries, notifies events, and waits for replies to queries;
- Interaction Controller: this module regulates the interactions with throwboxes and user nodes in order to retrieve data; it also dispatches to the content manager queries, either issued by the service container or by other nodes. Two sub-modules specialize specific functionalities: i) the sub-module *Interaction with user nodes* regulates the temporary storage to the user node data repository of data retrieved from sources in radio range and processes queries coming from other user nodes, ii) the sub-module *Interaction with throwboxes* is charge of issuing the storage/retrieval of data to throwboxes;
- Content Manager: it processes queries received from peer nodes and verifies whether the required information has to be retrieved (stored) from (to) the data repository onboard of user nodes;
- Network Interface: provides the basic network functionalities for one-hop communications.

As concerns throwboxes, we find:

- Interaction with user nodes: this module is in charge of processing the requests coming from the user nodes in radio range of the throwbox to either store or retrieve data; such requests are then forwarded to the throwbox primitive module;
- Throwbox Primitives: it issues the appropriate commands to the DHT system in order to handle data to (from) the distributed storage on the set of throwboxes;
- Network Interface: this module provides basic network functionalities for one-hop wireless communications;
- Distributed Hash Table (DHT): it is used to store data, where data indexing occurs via identifiers plus a locality based address of the DHT system[3].

In the following we describe in detail the primitives implemented by each of the modules described before, together with the main operational features.

4.1 User nodes

4.1.1 Service Container

The service container provides the necessary programming abstractions which allow services to utilize the R-P2P middleware. In particular, it executes services running inside user nodes, and provides facilities for services deployment, execution, deprecation and update. From the service controller, services are able to (i) query for data, issuing a `query` that is forwarded to the Interaction Controller (ii) generate and advertise contents that can be of interests for other users nodes. These functionalities are enabled through the following primitives:

- `search(query)` → `result`: this primitive is invoked by the Service Container towards the Interaction Controller, whenever a service, running on a user node, requires to retrieve some information specified by a certain `query`. If the requested information is already available in the device's storage, it is returned through the `result` variable as a `contentList` . If no local information is matching the specified `query`, a `no-data` response is returned through the `result` variable. In this second case, the underlying Interaction Controller invokes an appropriate callback primitive as soon as the queried content becomes available.
- `publish(advertisement)`: this primitive is invoked by the Service Container when some user/service-generated information is available. Such information takes the form of an `advertisement`, and is composed by its content and a topic (e.g., food, entertainment, shops, etc.), i.e. a meta-description of such content. As we will see in the following, the content is then stored in the Throwboxes storage system, while the topic only is diffused among user nodes.
- `notify()`: this primitive is invoked the Interaction Controller whenever a content matching a `query` previously issued by a service is found. In this case, the Interaction Controller notifies the Service Container of the retrieved `contentList`, and the Service Container then notifies this to the various services hosted. It is also possible that the Interaction Controller notifies an `advertisement` to the Service Container.

[2]We assume that suitable mechanisms are used in order to enforce cooperation

[3]DHT uses a standard wired or wireless network.

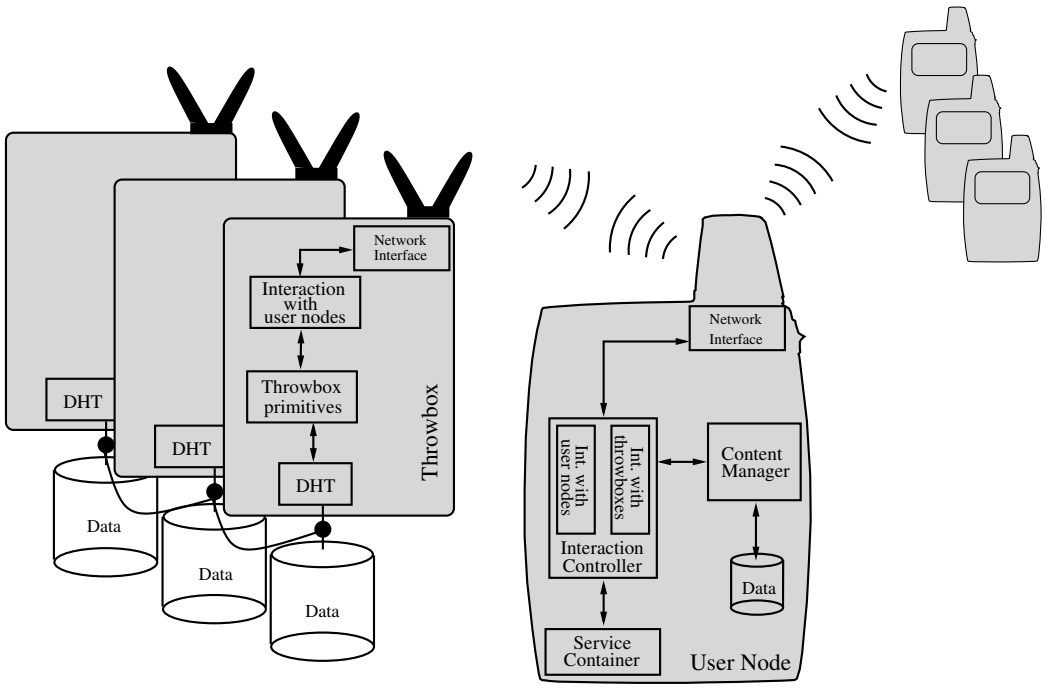


Figure 4: Functional blocks representation for the network components: throwboxes and user nodes.

Each `query` and `advertisement` issued by the Service Container is accompanied by some control information (*query-info* and *advertisement-info*) adding some additional information such as the spatial/temporal validity of the `query`/ `advertisement`(e.g. timeouts and/or a maximum distance for the forwarding of the `query`/`advertisement`). This prevents devices from propagating messages exceeding a given timeframe or outside a given spatial range.

4.1.2 Interaction Controller

The Interaction Controller "glues together" the various system components and regulates the interactions and flows of information inside the system. In particular, this module is in charge of processing the messages received from a user node or a throwbox: the module recognizes the incoming message as originated by a throwbox or a user node, and it then delegates other system components to process the request. In particular, it communicates with the Content Manager for resolving `query`s and retrieving/storing `contents`. Clearly, the Interaction Controller is linked to the Service Container for interfacing with the hosted services, thus invoking appropriate call backs whenever queried content or advertisements are received. Finally, as detailed in the following, it handles the interaction flow with both user nodes and throwboxes.

The data types used by this modules are the `queryList`, which is list of `query` items, and the `contentList`, which is a set of `content` items.

Interactions with user nodes

This submodule rules the interactions with user nodes. In particular, it provides the following primitives:

- `search(queryList)`→ `result`: this primitive triggers the transmission of a `search` message to user nodes in the communication range. Such message is composed as <`search`; `queryList`>, where `search` specifies the operation, while `queryList` the list of queries for which the user is asking content. In particular, the `queryList` will contain user node's own queries, for which content is possibly expected, and other users' queries, for which other users are expected to simply relay such `query`s. In case content is returned, this will be contained in the `result` variable;
- `send(contentList)`→ `result`: this primitive issues the transmission of the content to be sent to the peer node to fulfill some of the queries; basically it passes to that module the message <`send`; `contentList`>. It receives the `result` of the operation, e.g. `success` or `communicationfailure` in case a communication error is reported, the result is then passed to the Content Manager for dispatching data and in case a communication failure occurs, a basic control flow functionality processes retransmissions or drops the data transfer;
- `event(advertisement-info)`: this primitive sends the message <`advertisement-info`> to all the user nodes in radio range, thus notifying to interested user that a `publish`() primitive has been invoked by the Service Container some user node, with some content potentially interesting.

Interactions with throwboxes

This sub-module rules the interaction of a user node with a throwbox in radio range in order to store and/or retrieve data. We assume that identifiers of pending queries are passed to the sub-module by the Interaction Controller: the node own pending queries are matched with corresponding data on the throwbox; conversely, data corresponding to

Table 1: Summary of the data types that are used inside the system.

Name	Description	Modules
`query`	A `query` specifies the content that a service is requesting. A query is alway associated with (i) a unique query identifier `query-id`, which is determined by the node issuing the `query` (ii) a *query-info*, which contains side information such as the spatial/temporal validity of the query, and the ID of the node issuing the query.	Service Container
`content`	A `content` represents the information requested by a service through a query, or through the subscription to some interest.	Service Container, Content Manager
`advertisement`	An `advertisement` contains some user/service generated information. Any `advertisement` is accompanied by some control information, and in particular (i) a topic description (e.g., food, entertainment, shops, etc.) (ii) the advertisement body (e.g. a short text message) (iii) an identifier for the retrieval of associated data on the throwboxes.	Service Container, Interaction Controller
`queryList`	A `queryList` contains a set of `query`. Typically, a `queryList` includes queries issued by a a node itself, and queries issued by other nodes.	Service Container, Interaction Controller
`contentList`	A `contentList` contains a set of `content`items. This are sent as a response to a specific `query` issued by a user node.	Service Container, Interaction Controller

queries of other nodes are pushed to throwboxes. In particular, three primitives are implemented by the module:

- `store(content)`→ `result`: sends the message <`store`, `content`, `query-id`> to a throwbox in radio range to store `content`in the DHT, and associates it to the identifier `query-id`; then it waits for a return value reporting on the result of the operation (e.g., `success`, `alreadypresent` when another content is already associated to the same `query-id`, `communicationfailure` in case a communication error is reported);
- `get(query)`→ `content/exception`: it sends the message <`get`, `query-id`> to a throwbox in radio range; the effect is to retrieve from the DHT the `content` associated to the identifier `query-id`. This primitive waits for a return message with the `content` or an exception indication (e.g., content not yet/no more available, `communicationfailure`); it also removes `content` from the DHT.
- `get(advertisement)`→ `content/exception`: the same as `get(query)`, retrieves the `content` linked to `advertisement`.

Notice that, the user node exits the range of the throwbox, the exception `communicationfailure` is passed to the Content Manager in order rule the partial upload/download of a `content`.

4.1.3 Content Manager and Data

The content manager is in charge of the storage and deletion of `content`s from the local Data storage. Periodical refreshes permit to free storage: in particular `query`s, `advertisement`s and `content`s are associated with a limited spatial or/and temporal lifetime. The content manager compares periodically the information on the position of the node and the internal clock in order to free Data from `content`s out of scope. The Content Manager is accessible from other modules through the following primitives:

- `search(query)` → `contentList`: this query is used by the Interaction Controller for querying data that are possibly stored in the permanent repository.
- `store(content)`: this primitive is utilized for storing a `content` item in the permanent storage of the device.

4.1.4 Network Interface

This module implements some basic network functionalities above the Link Layer: it covers a primitive flow control and a basic node discovery. In particular, it regulates beaconing in order to detect nodes in radio range (the rate of beacon transmission influences on the probability of detecting nearby user nodes/throwboxes).

Based on the beacon information, the Network interface module advertises the node type, in order to discriminate either a throwbox or another user node. Once the beacon triggers a connection, this module is responsible for the binding mechanism of the current communication with the Interaction Controller.

Three main functions are implemented: a `send` primitive to pass messages to the MAC/LL layer together with a `fragment`/ `reassembly` primitive for the split and reconstruction of long chunks of transmitted and received data and a `notify` primitive which passes either received data or transmission reports to the Interaction Controller.

The Network Interface module implements a basic flow control mechanisms. In particular, when a large `contentList` is transmitted, the communication may end due to the expiration of the contact, e.g. nodes exited radio range; in such cases a notification message will be passed to the Interaction controller which will pass the list of `query-id`s successfully transmitted to the Content Manager: the corresponding `content` are erased from the local storage. The transmission of the `contentList` will be restored at the next contact with a throwbox. Also, when several devices are transmitting to a throwbox, a backpressure messaging from throwboxes can regulate concurrent upload flows.

5. THROWBOX NODES

5.1 Throwbox primitives

This module is the dual module of interaction with throwboxes present onboard of user nodes and implements two dual procedures `store` and `get`, plus a `take` procedure. This module is also in charge of interworking with the DHT by leveraging features provided by the DHT modules.

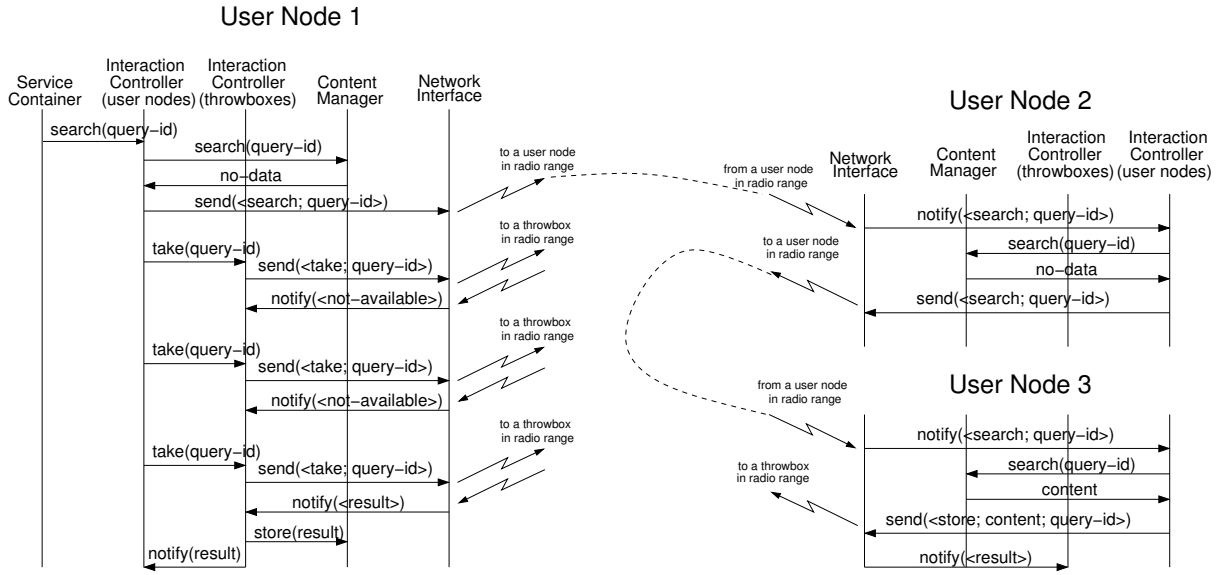


Figure 5: Interaction diagram for the sequence of operations involving user nodes.

`store(content)`→ `result`: it is used to store into the DHT the content; the indexing is performed by means of the identifier `content-id` and the result of the operation is returned (e.g., `success`, `alreadypresent` when another content with the same `content-id`exists, `DHT-failure`);

`get(query)`→ `content`/ `exception`: it requests to the DHT module to retrieve from the DHT the content indexed with the identifier `query`; if the content is not available, it returns an exception (e.g., content not yet/no more available);

`take(query)`→ `content`/`exception`: same as `get` but the data corresponding to the identifier `query` is removed from the DHT.

5.2 Network Interface

The functionalities of this component are similar to those of the peer module on user nodes.

5.3 DHT

This module implements the local functions for the DHT. We note that, in the proposed system, performance and scalability issues are more relevant than handling of joining/leaving of nodes since the the throwboxes belong to a static network. Chord [21] is our reference DHT implementation on the throwboxes; the choice follows since in R-P2P performance and scalability are more relevant that handling joining/leaving throwboxes.

DHT is in charge of storing, retrieving, and removing contents, indexed through identifiers. This module interacts with similar DHT modules deployed in other throwboxes, according to the interconnection topology adopted and implemented by the specific DHT solution.

To improve the performance of the system, contents should be stored by considering their locality: e.g., a content related to a given location L be stored in a throwbox deployed close to L. In order to fulfill this requirement while maintaining the strengths of DHTs (e.g., balancing node loads and storage usage, fault tolerance and recovery, scalability), the throwboxes are organized as a set of interconnected "localized" DHTs: all the throwboxes located in a given geographical area are part of the same DHT; a "localized" DHT is named through a "location" identifier, such as `downtown.trento.it`. Such identifiers are configuration parameters of the throwbox, and can be retrieved by user nodes.

The identifiers of stored contents (e.g., the answer to a query) are structured as <`location-id`, `content-id`>, where `location-id` is the location identifier of a "localized" DHT, and `content-id` is the identifier to be hashed by the DHT.

When a throwbox receives a message from a user node to `get`/`take`/`store` contents, the DHT module has to process it. As a first step, DHT module checks whether content's `location-id` is the one of the throwbox; if yes (due to the normal user node mobility, this is the most probable case), the DHT module hashes the `content-id` and performs the required operations to `get`/`take`/`store` the content. Otherwise, the DHT module accesses a global naming service (e.g., a DNS) for retrieving the address of a throwbox belonging to the remote "localized" DHT, identified by location-id, and forwards the request to it. The procedure to recover the DHT when a throwbox leaves a "localized" DHT (e.g., due to a crash) must be enriched in order to update the entries in the naming service.

In Fig 5 and Fig. 6 we reported the main interaction diagrams for R-P2P.

6. PRELIMINARY IMPLEMENTATION

We implemented a preliminary version of the R-P2P architecture over off-the-shelf components. In order to comply with the requirements deriving from the specific application scenario considered, we chose to rely mostly on commercially available software/hardware platforms. The architecture is composed by 2 parts: a mobile one, residing on users' personal handheld devices, and a fixed one, acting as the Throwboxes of the R-P2P architecture and interfacing with the fixed infrastructure. In the following, we provide a short overview of both components.

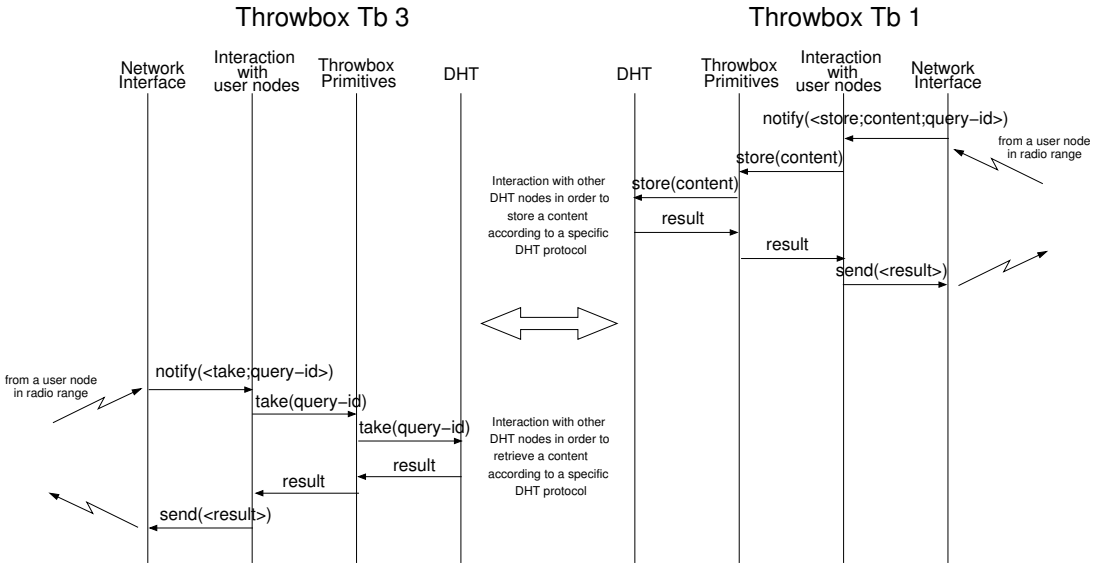


Figure 6: Interaction diagram for the sequence of operations involving throwboxes.

Tab. 2 presents a concise summary of the technologies and devices used in the R-P2P preliminary implementation.

6.1 U-Hopper: the mobile part of R-P2P

U-Hopper is a User-centric Heterogeneous Opportunistic Middleware [3] residing on mobile nodes. It exploits proximity communication interfaces (i.e., Bluetooth, Wi-Fi, etc.) in order to (i) gather localized information originating from sources embedded in the environment (ii) opportunistically disseminate queries from other user nodes (iii) deliver data to Throwboxes at any occasion. The content being exchanged by user nodes include data read from T-Nodes or queries generated by user nodes.
U-Hopper provides the necessary programming abstractions for implementation and deployment of R-P2P based services, and is composed, and includes various modules of the R-P2P architecture.

In order to embrace the largest number of "potentially available" mobile devices, we selected smartphones as the target platform to develop U-Hopper. In fact, smartphones are nowadays typically carried around by users during their daily activities; currently, also, they have reached a sufficiently large computing and communication power to perform very complex operations [4].

[4]As it will be clear in the following, the U-Hopper middleware can run also on other devices as long as the Java run time environment and some form of proximity communications are supported. Smartphones posses just the *minimum* set of hardware/software requirements to running the middleware platform.

HW platform	Nokia *E*65, Nokia N80, Dell600
OS	Symbian OS 9.1, Ubuntu 6.11
SW platform	J2ME (MIDP 2.0), J2SE 1.5
BT version	Bluetooth 1.2

Table 2: The R-P2P architecture implementation details in a nutshell.

We developed U-Hopper as a Java Midlet running over J2ME (MIDP profile 2.0), a widely diffused and standardized computing environment currently available on most smartphones shipped today.

We choose to rely on Bluetooth technology for achieving localized peer-to-peer data exchanges among mobile nodes. This is due to the large availability of this networking technology on mobile phones and to the available J2ME programming APIs (such as the well-known JSR 82 [1]. Obviously, Bluetooth is not properly designed for opportunistic communications, given the amount of time typically required for establishing a connection between two devices. To shorten up this connection time, we leveraged on some assumptions and on few properties of the Bluetooth technology. As an example, we implemented a device caching mechanism that inhibits consecutive meetings of nodes.

Permanent storage is obtained through the J2ME Record Management Store (RMS), which allows to store information as an arrays of bytes, and to retrieve data using easy-to-use matching methods. We have extended these basic functionalities, and implemented a J2ME query processors, which is able to interpret queries and return the specified data. An example of U-Hopper query is reported in Alg. 1.

Algorithm 1 Example of a U-Hopper content query.

```
1: SELECT time, location, other
2: FROM events.trento.bars
3: WHERE time > 21 AND date = '21/06/2008'
```

6.2 Throwboxes

Throwboxes functionalities are implemented as a Java application running on laptops. As for the U-Hopper implementation, we assumed Bluetooth as the proximity communication technology, utilizing the Avetana Bluetooth library as the Bluetooth stack implementation. This allows us to reuse much of the U-Hopper developed software.

In the current version of R-P2P, the Throwboxes imple-

mentation has been oversimplified: as a first step, in fact, we assumed Throwboxes to be connected to an IP network and to access to a shared centralized MySQL database. Through this database it is possible to (i) store queried content and (ii) store advertisements. Clearly, the spatial/temporal indexing of data is simplified. In the next phase, the DHT functionalities will be implemented replacing of the database with a full DHT solution and instantiating the related primitives accordingly.

The interested reader may refer to `u-hopper.create-net.org` for details and code downloads.

7. CONCLUSIONS

In this paper we described R-P2P, a system reproducing a request-response model for distributed queries. R-P2Pleverages a Delay-Tolerant Wireless Network: queries, tagged with an identifier, are diffused through opportunistic communications and responses are stored and retrieved (by using the query identifier), through interconnected throwboxes equipped with a set of DHTs.

The described solution does not account for the fact that the requested data might be already available on the throwboxes, e.g., because stored as an answer of some previous query. In this case a new query should not be issued, but data could be simply retrieved from the throwbox system. Future research will address this cases: a possible solution is to replace DHTs, now based on data indexing, with an unstructured P2P architecture, such GIA, Edutella, where the stored contents are labeled with keywords used to retrieve them. Thus, throwboxes would form an unstructured P2P storage network. Edutella [14] is a possible initial candidate: through metadata schema and attributes, defined in RDF, it would be possible to handle not only files, as in GIA, but also generic data and complex queries. The target will be to enhance the P2P system by forcing the Throwbox traffic to adhere to data localization.

8. REFERENCES

[1] JSR-000082 JavaTM APIs for Bluetooth.

[2] Carreras, I., Miorandi, D., and Chlamtac, I. A framework for opportunistic forwarding in disconnected networks. In *Proc. of Mobiquitous* (Palo Alto, USA, July 17–21, 2006).

[3] Carreras, I., Tacconi, D., and Miorandi, D. Data-centric information dissemination in opportunistic environments. In *MASS* (Pisa, Italy, October 2007).

[4] Chaintreau, A., Hui, P., Crowcroft, J., Diot, C., Gass, R., and Scott, J. Impact of human mobility on the design of opportunistic forwarding algorithms. In *Proc. of INFOCOM* (Barcelona, Spain, April 23–29, 2006).

[5] Fall, K. A delay-tolerant network architecture for challenged internets. In *Proc. of ACM SIGCOMM* (Karlsruhe, Germany, March 25–29, 2003).

[6] Khelil, A., Becker, C., Tian, J., and Rothermel, K. An epidemic model for information diffusion in MANETs. In *Proc. of ACM MSWiM* (Atlanta, Georgia, Sept. 28, 2002), pp. 54–60.

[7] Lau, W. H. O., Kumar, M., and Venkatesh, S. A cooperative cache architecture in support of caching multimedia objects in manets. In *WOWMOM* (New York, NY, USA, 2002), ACM, pp. 56–63.

[8] LeBrun, J., and Chuah, C. Bluetooth content distribution stations on public transit. In *MobiShare* (2006).

[9] Lee, U., Magistretti, E., Zhou, B., Gerla, M., Bellavista, P., and Corradi, A. MobEyes: smart mobs for urban monitoring with vehicular sensor networks. Tech. Rep. 060015, UCLA CSD, 2006.

[10] Leguay, J., Lindgren, A., Scott, J., Friedman, T., and Crowcroft, J. Opportunistic content distribution in a urban setting. In *Proc. of ACM Chants* (Florence, IT, September 15, 2006).

[11] Lindemann, C., and Waldhorst, O. P. A distributed search service for peer-to-peer file sharing in mobile applications. In *P2P* (Washington, DC, USA, 2002), IEEE Computer Society, p. 73.

[12] Luo, J., Hubaux, J.-P., and Eugster, P. T. Pan: providing reliable storage in mobile ad hoc networks with probabilistic quorum systems. In *MobiHoc* (New York, NY, USA, 2003), ACM, pp. 1–12.

[13] Malkhi, D., Reiter, M., and Wright, R. Probabilistic quorum systems. In *PODC* (New York, NY, USA, 1997), ACM, pp. 267–273.

[14] Nejdl, W., Wolf, B., and et al., C. Q. Edutella: a p2p networking infrastructure based on rdf. In *11th International Conference on World Wide Web* (2002).

[15] Ott, J. Application protocol design considerations for a mobile internet. In *MobiArch '06* (New York, NY, USA, 2006), ACM, pp. 75–80.

[16] Ott, J., and Pitkänen, M. Dtn-based content storage and retrieval. In *IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications* (Helsinki, 18-21 June 2007).

[17] Papadopouli, M., and Schulzrinne, H. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *ACM MobiHoc* (Long Beach, NY, 2001).

[18] Sailhan, F., and Issarny, V. Cooperative caching in ad hoc networks. In *MDM* (London, UK, 2003), Springer-Verlag, pp. 13–28.

[19] Shah, R., Roy, S., Jain, S., and Brunette, W. Data mules: Modeling a three-tier architecture for sparse sensor networks. In *IEEE SNPA Workshop* (May 2003).

[20] Spyropoulos, T., Psounis, K., and Raghavendra, C. S. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In *SIGCOMM WDTN* (2005), ACM.

[21] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw. 11*, 1 (2003), 17–32.

[22] Yang, G., Chen, L.-J., Sun, T., Zhou, B., and Gerla, M. Ad-hoc storage overlay system (asos): A delay-tolerant approach in manets. In *IEEE MASS* (Vancouver, October 2006).

[23] Yin, L., and Cao, G. Supporting cooperative caching in ad hoc networks. *IEEE Trans. on Mobile Computing 5*, 1 (2006), 77–89.