CarPal: interconnecting overlay networks for a community-driven shared mobility

Vincenzo Ciancaglini, Luigi Liquori, Laurent Vanni

CarPal: interconnecting overlay networks for a community-driven shared mobility*

Vincenzo Ciancaglini, Luigi Liquori, and Laurent Vanni

INRIA Sophia Antipolis Méditerranée, France
Email: firstName.lastName@sophia.inria.fr

Abstract. Car sharing and car pooling have proven to be an effective solution to reduce the amount of running vehicles by increasing the number of passengers per car amongst medium/big communities like schools or enterprises. However, the success of such practice relies on the community ability to effectively share and retrieve information about travelers and itineraries. Structured overlay networks such as Chord have emerged recently as a flexible solution to handle large amount of data without the use of high-end servers, in a decentralized manner. In this paper we present CarPal, a proof-of-concept for a mobility sharing application that leverages a Distributed Hash Table to allow a community of people to spontaneously share trip information without the costs of a centralized structure. The peer-to-peer architecture allows moreover the deployment on portable devices and opens new scenarios where trips and sharing requests can be updated in real time. Using an original protocol already developed that allows to interconnect different overlays/communities, the success rate (number of shared rides) can be boosted up thus increasing the effectiveness of our solution. Simulations results are shown to give a possible estimate of such effectiveness.

Keywords. Peer to peer, overlay networks, case study, information retrieval, car sharing

1 Introduction

1.1 Context

Car pooling is the shared use of a driver's personal car with one or more passengers, usually but not exclusively colleagues or friends, for commuting (usually small-medium recurring trips, like e.g. home to work or home to school). Amongst the many advantages it decreases traffic congestion and pollution, reduces trip expenses by alternating the use of the personal vehicle amongst different drivers and enables the use of dedicated lanes or reserved parking places where made available by countries aiming to reduce the global dependency of petrol.

Car sharing is a model of car rental for short periods of time (rather than the classical car rental companies), where a number of cars, often small and energy-efficient, are spread on a small territory, like a city. Customers first subscribe to a company who exploits and maintains the car park, then use those cars for their personal purposes. Service fees are normally per kilometer and insurance and fuel costs are included in the rates. Car sharing is an interesting option for families that need a second car but do not want to buy it. Modern geolocation technologies using GPS and mobile phones help to

* Supported by AEOLUS FP6-IST-15964-FET Proactive.

find the closest car to pick. The same economical/ecological advantage of car pooling applies, and mathematically speaking they are parameters of the same function we want to minimize.

1.2 Problem overview

In Car sharing/pooling services, an Information System (IS) has been showed to be essential to match the offers, the requests, and the resources. The Information System is, in most cases, a front-end web site connected with a back-end database. A classical client-server architecture is usually sufficient to manage those services. Users register their profile to one Information System and then post they offers/requests. In presence of multiple services, for technical and/or commercial reasons, it is not possible to share contents across the different providers, despite the evident advantage. As a simple example the reader can have a quick look on Equipage06 [**?**] and Otto and co [**?**], two websites concerning car pooling in the French Riviera region. At the moment the two do not communicate at all and do not share any user profile neither they share any request, even if they operate on the same territory and with the same objectives. Since both services are non profit, the reason for this has probably to be found in the client-server nature of both Information Systems that, by definition, are not designed to collaborate with each other. Although in principle this does not affect the correct behavior of both services, it is clear that interoperability between the two would increase the overall quality of the service. Moreover, the classical shortcomings of client-server architectures make both service unavailable in case both servers are down.

1.3 Contributions

As main contributions in this paper:

- we design and implement a peer-to-peer based Carpool information system, that we call *CarPal*: this service is suitable to be deployed with a very low infrastructure and can run on various devices spanning from PC to a small intelligent devices, like smartphones;
- we customize the Arigatoni protocol [CCL08] and its evolution, the Synapse protocol [LTV$^+$09], both specialized for resource discovery in overlay networks in order to allow two completely independent CarPal-based Information Systems to communicate without the need of merging one CarPal system into the other or, even worse, build a third CarPal system including both.

1.4 Outline

The rest of the paper is organized as follows: in Section 2, we introduce our CarPal service and we show how it is mapped onto a Distributed Hash Table. Section 3 describes the interconnection of different CarPal systems by means of the Synapse protocol developed in our team and tested over the Grid'5000 platform[1] In Section 4 we show a running example with a proof-of-concept that we have implemented in our team on the basis of a real case of study in our French Riviera area of Sophia Antipolis a technological pole of companies and research centers. A GUI is also presented[2]. Section 5 concludes and presents some further work.

[1] See `http://www-sop.inria.fr/teams/lognet/synapse`.
[2] See `http://www-sop.inria.fr/teams/lognet/carpal`.

2 Application architecture

2.1 Application principles

One of the most important features for a car share application is to be able to maximize the chances of finding a match between one driver and one or more travelers. From this comes the choice of arranging the database by communities in order to put in touch people who most likely share the same traveling patterns in space and time (e.g. work for the same company, attend the same university and so on). Another important aspect is to be able to update the planned itinerary information as quick as possible so that a last minute change in plans can be easily managed and updated and may eventually lead in finding a new match.

For the above reasons, CarPal has been intended as a desktop and mobile application running on a peer-to-peer overlay network. This allows a community of people to spontaneously create its own travel DB (which, as it will be shown later, can be interconnected with siblings communities) and manage it in a distributed manner. Moreover, it constitutes a flexible infrastructure where,by being deployed on connected mobile devices, it will be possible to develop more advanced info-mobility solutions that might take into account the position of the user/vehicle (via the internal GPS), geographically-aware network discovery or easy network join or vehicle tracking through checkpoints with the use of Near Field Communications technologies [NFC].

2.2 CarPal in a nutshell

The working principle is simple: a user running a CarPal client on his mobile or desktop will connect to one or more communities to which he is member (i.e. he has been invited or his request has been accepted). Two operations are then available, namely (i) publishing a new itinerary and (ii) finding a matching itinerary.

Publishing a new itinerary. When a CarPal user has a one time or recurring trip that he wants to optimize cost-wise he can publish his route in the community in hope to find someone looking for a place in the same route and time window to share the ride with. A planned itinerary is usually composed by the following data:

- *Trip date and number of repetitions* in case is a recurring trip;
- *Departure place and arrival place*, whose representation is critical since a too high granularity might lead to miss similar results;
- *Departure time*;
- *Arrival time* or at least an estimate given by the user;
- *Number of available seats* to be updated when another passenger asks for a place;
- *Contact*, usually an e-mail or a telephone number;
- Further useful information, i.e. animal allergies, women-only car etc.;

Moreover, from a functional point of view, a trip e.g. from a place A to a place D may include several checkpoints, meaning that the user offering his car can specify one or more intermediate stops in the itinerary where he is willing to pick up or leave a passenger.

Once the user has inserted all the needed data (date, departure, arrival, times, seats and optional checkpoints), the trip is decomposed in all the possible combinations: for example, a trip containing the legs A-B-C-D (where B and C are checkpoints

specified by the user) will generate the combinations A-B, A-C, A-D, B-C, B-D and C-D. This operation is commonly known as *Slice and Dice*. Since the number of possible combinations can increase exponentially with the number of checkpoints, there is a software limitation to 3 maximum stops in the trip.

Each combination is then stored in the DHT as an individual segment; moreover all the segments who don't start form A are marked as estimated in departure time since, given a trip made of different checkpoints, only the effective departure time can be considered reliable, the others being subject to traffic conditions and contingencies. Geographic and time information must be encoded in such a way to be precise enough to be still relevant for our purposes (someone leaving from the same city but 10 km far is not a useful match) yet broad in a way that a punctual query will not skip important results.

Every checkpoint (including departure and arrival point) can either be inserted directly as geographical coordinates (using the GPS capabilities of modern mobile devices) or as an address that will then be converted in geographical coordinates using Reverse Geolocation APIs made available by services such as Google Maps [Goo]. Such coordinates will then be rounded in the hash key encoding in order to group together locations within a given radius (around 5 kilometers). Concerning time approximation, a 20 minutes window is used instead to approximate departure times. Both during an insertion or a query, anything within the 0-19 minutes interval will be automatically set at 10 minutes, 20-39 will be set at 30 minutes and 40-59 at 50.

Finding a matching itinerary and one seat. A user wishing to find a ride can perform a search by inserting the following information:

- *Date* of the trip;
- *Departure* place and time (picked on a map between the proposed points;
- *Arrival* place and wished time, picked in the same manner as the departure.

To increase the chances of finding a match, only part of the search criteria can be specified, allowing e.g. to browse for all the trip leading to the airport in a certain day disregarding the departure time (giving the user the chance of finding someone leaving the hour before) or the departure point (giving the user, in case of nobody leaving from the same place as him, to find someone leaving nearby to join with other means of transportation). Moreover it is possible to specify checkpoints in the search criteria too, in order to have the system look for multiple segments and create aggregated responses out of publications from multiple users.

Negotiation. Once the itinerary has been found it will be possible to contact the driver in order to negotiate and reserve a seat. If the trip is an aggregation of different drivers' segments all of them will be notified through the application. The individual trip records will then be updated by decreasing the available seats number.

2.3 Encoding CarPal in a DHT

The segments are stored in the DHT according to Table 1. Since we are not able yet to perform useful range queries on the DHT, multiple keys are inserted or updated for each entry, representing sets of trips grouped according to different criteria:

1. Is the actual trip record, associated to a unique TRIP_ID, that will be updated, for example, when someone books a seat. The information stored concerns trip date

Criteria	Key	Value	Trip Grouped by
1	TR|TRIP_ID	♣	Individual
2	T|DATE|DEP|TOD|ARR|TOA	list[TRIP_ID]	Dep & Arr & Time
3	DA|DATE|DEP|ARR	list[TRIP_ID]	Dep & Arr
4	D|DATE|DEP	list[TRIP_ID]	Dep
5	A|DATE|ARR	list[TRIP_ID]	Arr
6	U|USER_ID	list[TRIP_ID]	User id

where ♣ = [DATE,DEP,TOD,ARR,TOA,SEATS,REF,PUB]

Table 1. Different data structures stored in the DHT for each entry

DATE, departure place DEP and time TOD, arrival place ARR and time TOA number of available seats SEATS (or cargo space, in case of shared goods transportation), a reference to contact the driver REF, and if the trip has to be public or not PUB. Depending on the needs more information can be appended to this record;

2. Represents a set of trips having the same date, departure/arrival point and departure/arrival time. The key is created by concatenating the token T, trip date DATE, departure place DEP, departure time TOD, arrival place ARR and arrival time TOA. As value is the list of TRIP_ID pointing to the trip records. The key is created by appending the token TR to the TRIP_ID;
3. Is a set of trips grouped by date, departure and arrival place. It will be used to query in one request all the trips of the day on a certain itinerary. The key to store them in the DHT is consequently made by appending thhttp://www.technoholik.com/news/ten-reasons-why-we-are-excited-about-notion-ink-adam/e token DA, trip date, departure and arrival point;

4-5. Are two sets of trips arranged by day and by departure or arrival point. The key is therefore made by concatenating either the token D (for departure) or A (for arrival) to the DATE and departure or arrival point DEP or ARR. This key can be used e.g.to query in one request all the trips of the day leaving from a company or all the trips of the day heading to the airport;

6. Is a set of trips for a given user. The key is therefore the token U prepending the USER_ID itself.

2.4 Network architecture

The overlay chosen for the proof of concept is Chord [SMK+01] although other protocols could be used to exploit the locality of the application or a more direct geographical mapping (see Section 5.2). Even on a simple Chord several mechanisms to ensure fault tolerance can be put in place, like data replication using multiple hash keys or request caching. To allow a new community to be start up, a *public tracker* has been put in place on the Internet. The public tracker is a server whose tasks can be resumed as following:

- It allows the setup of a new community by registering the IP of certain reliable peers, in a YOID-like fashion [Fra00];
- It acts as a central database of all the communities, keeping track of them and their geographical position;

– consequently, it can propose nearby overlays to improve the matches by placing co-located peers;
– It acts as a third party for the invitation of new peers into an overlay;
– It can provide statistical data about the activity of an overlay, letting a user know if a certain community has been active lately (and thus if it is worth joining);
– It acts as the entry point to download the application and get updates.

3 Interconnecting different communities

3.1 Context and Motivations

As previously stated, CarPal has been conceived as a service where new communities of Car poolers can be put in place without the need of an existing IT infrastructure (e.g. a dedicated online service to join like [**?**] and [**?**]).

– As a first effect we would expect to see is the birth and growing of different overlays around communities sharing the same interests, activities, jobs and in general anything that could lead to a common travel pattern (e.g. company employees, universities personnel, sports club members).
– Another expected consequence however will be to have multiple CarPal communities *geographically overlapping* i.e. residing in the same area and *not being aware of each other* thus one not taking advantage of the other's offerings. Often companies are very close geographically and they have the same working timetable. If nearby communities put in place different CarPal overlays, those possible matches will not be taken into account.

Under certain conditions in order to overcome such a limitation a search operation for a given itinerary within a community can be extended to other overlays being geographically close.

3.2 Query extension to nearby communities

A request for an itinerary can be routed through co-located nodes who are members of different CarPal overlays. The interconnection of a node to overlays other than his original is established via a social mechanism, where a user can ask or receive and invitation to join other communities. Since every community shares the same structure for the hash key the node will then be able to query all his communities and act as well as a proxy for any request going through it. Moreover, as mentioned in 5, it will be possible to have a node interacting even with existing online services such as [**?**] and [**?**]. the query extension mechanism is implemented using the Synapse protocol developed in our team and described in details in [LTV+09]. We hereby present a little summary of its capabilities.

3.3 Synapse in a nutshell

The protocol is based on co-located nodes, also called *Synapses*, serving as low-cost natural candidates for inter-overlay bridges. In the simplest case (where overlays to be interconnected are ready to adapt their protocols to the requirements of

interconnection), every message received by a co-located node can be forwarded to other overlays the node belongs to. In other words, upon receipt of a search query, in addition to its forwarding to the next hop in the current overlay (according to their routing policy), the node can possibly start a new search, according to some given strategy, in some or all other overlay networks it belongs to. This obviously implies to providing a Time-To-Live value and detection of already processed queries, to avoid infinite loop in the networks, as in unstructured peer-to-peer systems. Applications of top of Synapse can see those inter-overlay as a unique overlay.

In case of concurrent overlay networks, inter-overlay routing becomes harder, as intra-overlays are provided as some black boxes: a *control* overlay-network made of co-located nodes maps one hashed key from one overlay into the original key that, in turn, will be hashed and routed in other overlays in which the co-located node belongs to. This extra structure is unavoidable to route queries along closed overlays and to prevent routing loops.

3.4 Synapse performances and exhaustiveness

Our experiments and simulations show that a small number of well-connected synapses is sufficient in order to achieve almost exhaustive searches in a set of structured overlay networks interconnected together.

In order to test our inter-overlay protocol on real platforms, we have initially developed JSynapse, a Java prototype that fully implements a Chord-based inter-overlay network. We have experimented with JSynapse on the Grid'5000 platform connecting more than 20 clusters on 9 different sites. Again, Chord was used as the intra-overlay protocol. The created Synapse network was first made of up to 50 processors uniformly distributed among 3 Chord intra-overlays. Then, still on the same cluster, as nodes are quad-core, we deployed up to 3 logical nodes by processor, thus creating a 150 nodes overlay network, nodes being dispatched uniformly over 6 overlays. During the deployment, overlays were progressively bridged by synapses (the degree of which was always 2).

Figure 1 (left) shows the satisfaction ratio when increasing the number of synapses (for both white and black box versions). A quasi-exhaustiveness is achieved, with only a connectivity of 2 overlays per synapse. Figure 1 (right) illustrates the very low latency (a few milliseconds) experienced by the user when launching a request, even when a lot of synapses may generate a lot of messages. Obviously, this result has to be considered while keeping the performances of the underlying hardware and network used in mind. However, this suggests the viability of our protocols, the confirmation of simulation results, and the efficiency of the software developed.

3.5 Implementation in CarPal

For our scope we decided to adopt the so called *Black box* version of the Synapse protocol. The difference with the original Synapse approach is that, being aimed to route into non collaborative networks, instead of embedding the additional data needed for the inter-overlay routing in the request packets themselves, it actually uses a parallel Control Network. The reasons for this design choice, disregarding the fact that every CarPal overlay would be collaborative by definition being participated only by CarPal peers, are the follwing:

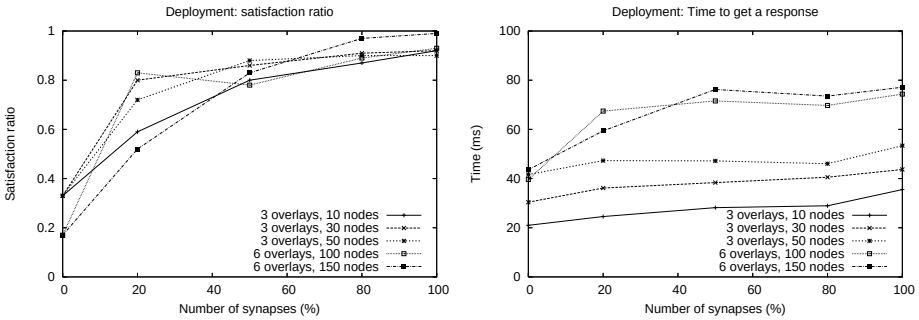

Fig. 1. Deploying Synapse : Exhaustiveness (left) and Latency (right)

- it offers the possibility to query an existing online service as if it was a non collaborative network, by having one or more synapses acting as clients
- it allows more control over the inter-overlay routing, by offering the possibility to perform selective flooding of only specific networks.

To achieve this the Control Network handles 2 different data structures (a KeyTable and a CacheTable). Both are implemented as Distributed Hash Tables on a global overlay participated by every CarPal node.

- **The Key Table** is responsible for storing the unhashed keys circulating in the underlying overlays. When a synapse-enabled peer performs a GET that has to be replicated in other networks, it makes the unhashed key available to the other synapses through the Key Table. The key K is stored using an index formed by a networks identifier as a prefix and the hashed key itself as a suffix. This way when a synapse on the overlay with e.g. ID = A will have to replicate e.g. H(K) = 123, it will be able to retrieve, if available, the unhashed key K from the KeyTable by performing a get of A|123.
- **The Cache Table** is used to implement the replication of get requests, cache multiple responses and control the flooding of foreign networks. It stores entries in the form of [H(KEY),TTL,[NETID],[CACHE]]. In a nutshell: NETID are optional and used to perform selective flooding on specific networks. When another synapse receives a GET requests, it checks if there is an entry in the Key Table (to retrieve the unencrypted key), and an entry in the Cache Table; if so, it replicates the GET in the [NETID] networks he is connected to, or in all his networks if no [NETID] are specified. All the responses are stored in the [CACHE] and only one is forwarded back, in order not to flood other nodes having performed the same request. A TTL is specified to manage the cache expiration and block the flooding of networks. When the synapse originating the request receives the first response, it can retrieve from the Cache Table the rest of the results. The cached responses should be sent back with the associated NETID. This can allow a with time to define a strategy of selective flooding to the networks who are better responding to a synapse request.

The **inter-overlay routing** takes place when a synapse peer wish to perform an extended query: before routing the request in his own community it adds an entry in the KeyTable, containing the unhashed key to be searched, and an empty entry in the CacheTable. When another synapse in the first overlay receives the request,

it looks for the unhashed key in the KeyTable and the corresponding entry in the CacheTable. If those are found, the co-located synapse will query for the same key in all his communities and store the results in the CacheTable, in order not to pollute the originating network with too many results. The requesting peer in the first network will then collect the results from the CacheTable upon reception of the first response.
Controlling the data. Since different CarPal overlays use different hash functions to map their keys a first level of privacy and control is guaranteed in case a community wish to have some control over the visibility of their information. At the present state there are 2 possible scenarios for accessing the data:

- A user can search for trips marked as both public and private in every overlay he's directly connected to. As previously stated, the connection to an overlay happens via invitation through a mechanism similar to certain social networks;
- If certain nodes of his own networks are member of other overlays, they can act as synapses and route queries from one network to another. However only the trips marked as "public" will be made available to a foreign request.

4 A Running example

We hereby present a first proof-of-concept for a CarPal application implementing the concepts exposed above. The software is still at an initial development but it has already been proven to be working in posting new routes and querying them across multiple networks. A basic user interface is proposed showing a first attempt to integrate a mapping service (namely, Google Maps [Goo]) in the application to render the user experience more pleasant and efficient, although no GPS capabilities and no reverse geolocation are in place yet.

4.1 Building the scenario

Let's see a practical example to better explain the logic behind the application. As a real world scenario for our proof-of-concept we chose the area of Sophia Antipolis in the department of Provence-Alpes-Cote d'Azur, France. The area (Figure 2 left) constitutes an ideal study case, being a technological pole with a high concentration of IT industries and research centers, thus providing several potential communities of people working in the same area and living in nearby towns (Antibes, Nice, Cagnes sur Mer to name some).

An engineer working in the area and willing to do some car pooling in order to reduce his daily transfer costs can publish his usual route to the CarPal overlay specific to his company. We assume the network has been already put in place spontaneously by him or some colleague of his. He can then use the CarPal application to publish his route with an intermediate checkpoint (as shown in Figure 3). As previously described, there is a checkpoint where our user is willing to stop and pick up some passengers.

4.2 Slice and Dice and encoding in the DHT

Starting from the above data all the possible combinations are generated leading to the segments shown in Figure 2 (right). Only the differences are reported, each of those segments share the same date, available seats and contact information. The 3

Trip date	15/01/2010
Departure	Nice
Departure Time	8.00
Checkpoint	Cagnes sur Mer
Checkpoint Time	8.30
Arrival	Sophia Antipolis
Arrival Time	9.00
Seats available	4
Contact	jsmith@email.com

Nice-Sophia	8.00-9.00
Nice-Cagnes sur Mer	8.00-8.30
Cagnes sur Mer-Sophia	8.30-9.00

Fig. 2. The geographical set-up (left), journey data (right) and sliced & diced segments (bottom right)

Fig. 3. CarPal application publishing a new trip

segments are then stored in the DHT by updating the appropriate keys or adding new ones, as shown in Table 2. Time and dates are converted to appropriate strings while geographical positions are identified by a placeholder (i.e. NICE, SOPH...).

A PUT operation represents the insertion of a new key not yet existing whereas the APPEND operation assumes that the key might be already in the DHT, in which case the value is simply updated by adding the new entry to the list. After the insertion the trip is published and available to be searched. From Figure 3 we can see that it's possible to set as an option that the trip will stay private. In that case, the segments will be discoverable only by peers members of the same network.

Criteria (see Table 1)	Operation	Key	Value
1	PUT	TR 123	♣
1	PUT	TR 124	♠
1	PUT	TR 125	■
2	APPEND	T 20100115 NICE 0800 SOPH 0900	123
2	APPEND	T 20100115 NICE 0800 CAGN 0830	124
2	APPEND	T 20100115 CAGN 0830 SOPH 0900	125
3	APPEND	DA 20100115 NICE SOPH	123
3	APPEND	DA 20100115 NICE CAGN	124
3	APPEND	DA 20100115 CAGN SOPH	125
4	APPEND	D 20100115 NICE	123
4	APPEND	D 20100115 NICE	124
4	APPEND	D 20100115 CAGN	125
5	APPEND	A 20100115 SOPH	123
5	APPEND	A 20100115 CAGN	124
5	APPEND	A 20100115 SOPH	125
6	APPEND	U jsmith@email.com	[123,124,125]

where ♣ = [20100115, NICE, 0800, SOPH,0900, 3, jsmith@email.com, public=true]
where ♠ = [20100115, NICE, 0800, CAGN,0830,3, jsmith@email.com, public=true]
where ■ = [20100115, CAGN, 0830, SOPH, 0900, 3, jsmith@email.com, public=true]

Table 2. DHT operations

4.3 Searching for a trip

A search for a trip happens in a similar way as the trip submission. As we can see in Figure 4 (left) the user can specify an itinerary, a specific time and even some intermediate segments to find all the possible combinations. Depending on the search criteria specified, the application will perform a query for either a key made of Time of Departure and Time of Arrival, for a more exact match, a key with only Departure and Arrival to browse through the day's trips or a key with only Departure or Arrival point for a broader search. Thanks to the Slice and Dice operation it is possible to aggregate segments coming from different users as Figure 4 (right) shows.

This way the driver has more possibilities to find guests in his car. Despite that there can still be some places available for his daily route. To optimize even further he might share his information with, for example, students of nearby universities with their own carpool network (that has the same functions and technology).

By marking his published itinerary as public, a member of the Enterprise Network allow the students to get matching results via a synapse (Figure 5), i.e. somebody registered to both networks (Figure 6). This allows the system to increase the chances of finding an appropriate match while maintaining good locality properties.

5 Conclusion and further work

There are several potential improvements, amongst which the following.

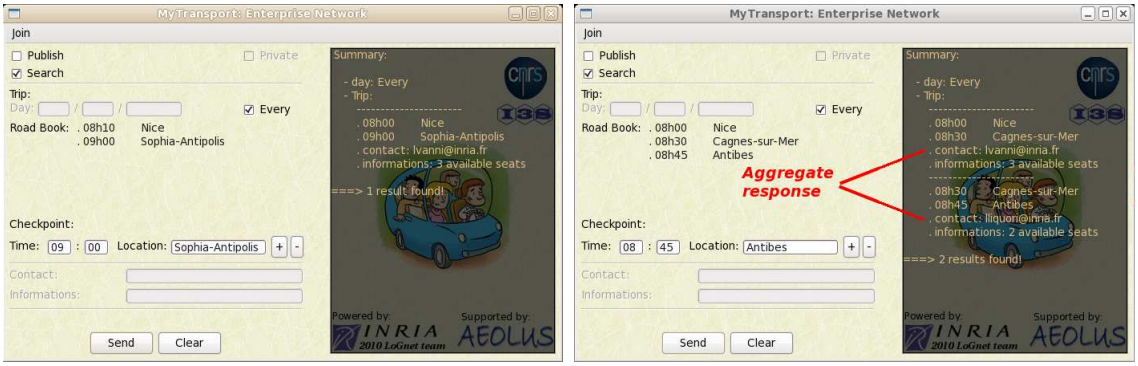

Fig. 4. Simple search vs. aggregate results

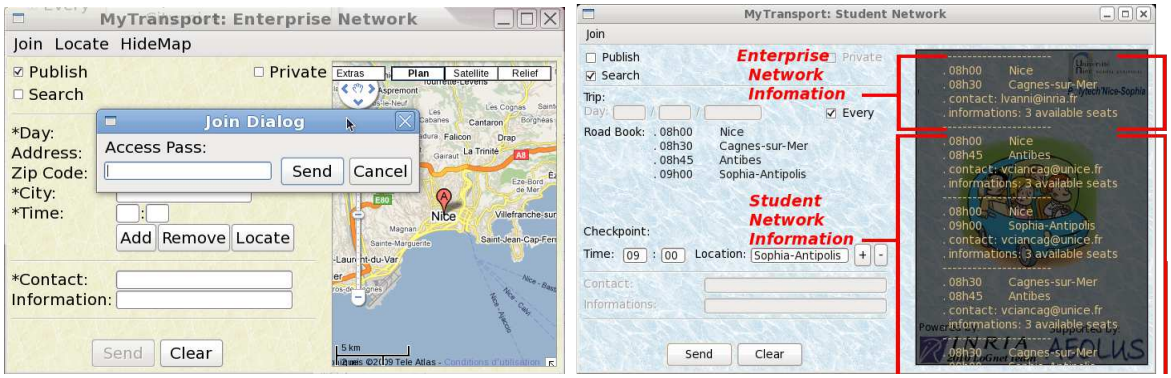

Fig. 5. Synapse creation (left). CarPal Students accessing result from Enterprise Network (Right)

5.1 Improved network bootstrap and community discovery

At the present state a new community can be setup or joint by passing through the tracker. This keeps track of community activities, their location, handles the join negotiation and restrictions and can suggests nearby communities that could be joined; however it constitutes also a centralized point of failure for all the communities. To improve further more the mechanism the following solutions can be put in place:

- Assuming that a community/overlay could very likely reside on the same network infrastructure (i.e. the enterprise intranet) a discovery protocol can be put in place leveraging existing technologies like Avahi [Ava] to discover new peers or new networks to join;
- Peer caching could be used to reconnect to previously connected peers whose activity is known to be reliable;
- An invitation to a new community could be handled physically via an Near Field Communications transaction [NFC]. A user with an NFC enabled phone could be invited by another user by simply swiping the phones together or touching a radio

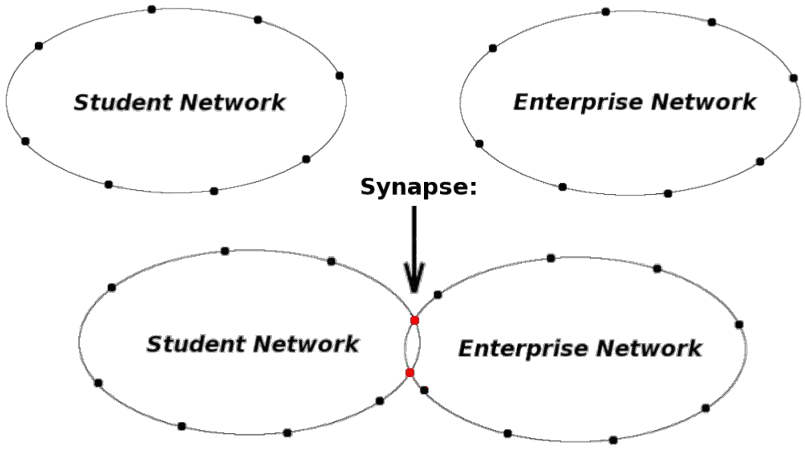

Fig. 6. Students, Enterprise and Synapsed Overlay Networks

tag. Furthermore this could be an additional guarantee of a user "reliability" as the new participant needs to be known and met by an existing member;
– The community database could be as well stored in the DHT itself, meaning by this that new communities could simply be discovered through specific requests routed through existing synapses to other networks in a ping-like way.

5.2 Semantic queries and specialized protocols

It appears clear that the current approach suffers from the limitation of a simple key-value approach. Such approach does not fit well into an application that finds her strength in the possibility of performing searches according to many different criteria. The adoption of a semantic hash function (such as [SH09]) would allow to cluster in nearby peers information semantically close (i.e. trips heading to sibling destinations or taking place in the same time window). Needless to say, with such hashing in place the adoption of an overlay protocol more suited to range queries (like P-Ring [CLM+07], P-Grid [ACMD+03] or Skipnet [NHD+03]) might lead the semantically significant range queries, where, for example, departure and arrivals can be geographically mapped and queried with a certain range in Km.

Another possible improvement (currently under study) would be to use a DHT protocol more suited for geo-located information. CAN [RFH+01]) in a 2D configuration is a first example of how this could be achieved. Mapping CAN's Cartesian space over a limited geographic area (like in Placelab [CRR+05]) could ease the query routing and eventually provide some strategic points to place synapsing nodes.

5.3 Overlay-underlay mapping optimizations

the overlay-underlay network mapping to avoid critical latency issues due to the fact that one logical hop can correspond to many physical hops. The issue is under investigation and could involve e.g. the use of several always-on peers to triangulate the "position"

of a peer over the Internet (according to latency metrics) and cluster together nearby peers (whereas nearby will mean sharing similar latencies to the same given references). Another issue would be to make the service firewall-resilient by implementing TCP Punch-hole techniques in the peer engine and in the tracker.

5.4 Backward compatibility with other Carpool services

To take into account issues like access to non collaborative networks or backward compatibility the Synapse protocol allows also a so called *black box* variant, whose first implementation is described in 3.5, that is suitable to interconnect overlays that, for different reasons, are not collaborative at all. This means that they only route packets according to their proprietary and immutable protocol. Being the black box more of a meta-protocol running on top of existing, and not necessarily peer-to-peer, structures, we can imagine strategically placed Synapse nodes being responsible of querying existing web services and returning the corresponding information as if they were coming from a foreign network. This open new scenarios were multimodality is easily integrated and made available for the nearby communities. The system needs to be properly designed in order to avoid a situation where too many peers act as a Distributed Denial Of Service, but the current infrastructure make it easily feasible.

5.5 User rating, social feedback

Being CarPal an application based on user-generated content and designed to put in touch people not necessarily acquainted to each other, it is important to implement some social feedback and security mechanism to promote proactive and good behaviors by the users. Borrowing from today's most successful web applications, 2 solutions can be imagined:

- A user rating should be put in place in order, for example, to allow passengers to evaluate a driver's punctuality, behavior and driving skills, and vice-versa. This feedback, similar to what systems like Ebay [**?**] already have since several years, can help maintaining a high level of quality in the service by giving an immediate picture of a driver or passenger's reliability;
- Some points can be assigned to users based on their activity in the community. The more a user will be proactive by publishing or subscribing to new trips in an overlay, the more "karma points" he will receive. A similar approach can be verified in Social News website like Digg [Dig] or Reddit [Red] and has become pretty common in today's social media. With the deployment and integration of new distributed services, these points could act as a "virtual cash" and grant access to features normally reserved to paying customers, thus motivating drivers and passengers to keep a community alive.

5.6 Other potential applications

The Car sharing/pooling is not the exclusive applicative field the overlay network technology we designed; with the same final objective of minimizing traffic, pollution and energy a service interconnecting transportation companies Information Systems could be envisaged. A "BoxPal" system could be easily build using the same overlay network technology: the only difference being the (more difficult) 3D bin-packing combinatorial algorithms employed instead of simple matching of drivers/cars/itinerary/car places.

References

[ACMD+03] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.

[Ava] Avahi project website. `http://avahi.org/`.

[CCL08] R. Chand, M. Cosnard, and L. Liquori. Powerful resource discovery for Arigatoni overlay network. *Future Generation Computer Systems*, 1(21):31–38, 2008.

[CLM+07] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke, and J. Shanmugasundaram. P-ring: an efficient and robust p2p range index structure. In *In prof. of SIGMOD*, pages 223–234, New York, NY, USA, 2007. ACM.

[CRR+05] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A case study in building layered dht applications. In *In Proc. of SIGCOMM*, pages 97–108, New York, NY, USA, 2005. ACM.

[Dig] Digg website. `http://www.digg.com/`.

[Fra00] P. Francis. Yoid: Extending the internet multicast architecture. Technical report, AT&T Center for Internet Research at ICSI (ACIRI), 2000.

[Goo] Google maps website. `http://maps.google.com`.

[LC07] L. Liquori and M. Cosnard. Logical Networks: Towards Foundations for Programmable Overlay Networks and Overlay Computing Systems. In *TGC*, volume 4912 of *LNCS*, pages 90–107. Springer, 2007.

[LTB09] L. Liquori, C. Tedeschi, and F. Bongiovanni. BabelChord: a Social Tower of DHT-Based Overlay Networks. In *In Proc. of IEEE ISCC*. IEEE, 2009.

[LTV+09] L. Liquori, C. Tedeschi, L. Vanni, F. Bongiovanni, V. Ciancaglini, and B. Marinkovic. Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks. Research report, INRIA, 2009. submitted, available on request, see also `http://www-sop.inria.fr/lognet/synapse/`.

[NFC] NFC forum website. `http://www.nfc-forum.org/`.

[NHD+03] J.A. Nicholas, J. Harvey, J. Dunagan, M.B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties, 2003.

[Red] Reddit website. `http://www.reddit.com/`.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Adressable Network. In *ACM SIGCOMM*, 2001.

[SH09] R. Salakhutdinov and G. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup service for Internet Applications. In *ACM SIGCOMM*, pages 149–160, 2001.