Towards a common architecture to interconnect heterogeneous overlay networks

Vincenzo Ciancaglini Luigi Liquori Giang Ngo Hoang
INRIA Sophia Antipolis Méditerranée
France
Email: name.surname@inria.fr

***Abstract*—This paper presents a novel overlay architecture to allow the design and development of distributed applications based on multiple interconnected overlay networks. Message routing between overlays is achieved via co-located nodes, i.e. nodes that are part of multiple overlay networks at the same time. Co-located nodes, playing the role of distributed gateways, allow a message to reach a wider set of peers while overlay maintenance remains localized to individual overlays of smaller size. To increase robustness, gateway nodes route messages in an unstructured fashion, and can discover each other by analyzing the overlay traffic. The approach is able to work in both "collaborative" scenarios, where overlay protocol messages can be modified to include additional inter-routing information, or non-collaborative ones. This allows for the interaction with existing overlay protocols already deployed.**

Keywords: peer-to-peer, overaly networks, distributed hash tables, overlay interconnection

I. Introduction

A. Context

Overlay networks (structured and unstructured) have been broadly recognized as a viable solution for the implementation of distributed applications of various nature: Distributed Hash Tables (DHT), application level multicast protocols, distributed object lookup, file systems, etc. In a real world scenario, a fully distributed design should rely as much as possible on regular, desktop-class machines, without the assumption of permanent connectivity, or even a public IP address. Although most of the protocols designed in recent years show interesting properties of scalability, fault tolerance and handling of dynamic topologies, studies such as [1] have shown that, under real networking conditions, an overlay can suffer severe performance degradation when deployed on a larger scale. This problem has been addressed in works such as [2] for specific cases, or, for a more general case, in Hierarchical Overlays (see Section II-C), in an effort to increase locality and reduce the size of individual overlays. A second challenge one is faced with when designing distributed applications is brought on by the fact that load balancing and data complexity are often inversely proportional. In the case of Distributed Hash Tables (DHTs), adding semantics to overlay keys or using non-consistent hash functions (such as locality preserving hash) for the addressing implies altering the data partitioning scheme, resulting in an uneven distribution of data across peers. Because of this, query complexity in distributed applications remains low. Works such as [3] or [4] show the implementation of complex queries on top of structured overlays, but their apporach is, again, specific to a data domain (in the abovementioned case, semantic and "geographical" data). What if an application wanted to exploit both? A third concern stems from security and anonimity issues due to the distributed nature of the system. Malicious attacks such as Sybil or Eclipse [5] can be a source of concern when one is designing a distributed system. Furthermore, due to the peer-to-peer message routing nature of such a protocol, it becomes possible for a malicious node to infer the activity of a peer by analyzing the ongoing traffic. Finally, there exists a more general problem regarding the cooperation of various systems. Existing systems are currently not able to communicate and exchange data with each other. As an example, the BitTorrent protocol [6] can use a DHT, based on the Kademlia protocol [7], to perform peer discovery. At the present state, there exist two different implementation of such a protocol (Mainline DHT and Azureus), and these two implementations are incompatible with one another.

B. Synapse interconnection protocol

With these premises in mind, we propose a novel architecture, henceforth referred to as *Synapse*, to allow for the transparent interaction between heterogeneous overlay networks. The general idea behind Synapse is to exploit co-located nodes, i.e. nodes connected to different overlays at the same time, as a form of distributed gateways capable of inter-routing queries and messages beyond one overlay. Nodes in an overlay can route messages to the overlay itself, using the correspondent protocol, or to foreign networks by contacting, in an unstructured way, previously discovered gateway nodes. Node discovery can be performed in an opportunistic way, by embedding additional information in the overlays messages, or independently via a peer exchange mechanism. The latter allows for a Synapse node to be completely independent from the underlying overlay protocol, thus giving the possibility of having existing overlay networks interact with each other without breaking network compatibility. A first version of the protocol has been presented in [22], of which the present work aims to be a generalization and an extension to multiple cooperation scenarios.

C. Summary

The rest of the paper is structured as follows: in Section II we present the previous work concerning overlay federation and collaboration. In Section III we lay out in detail the Synapse architecture, as well as the routing and node discovery mechanisms. Section IV briefly presents some details of the implementation in the OverSim simulator. Section V shows promising results gathered from simulations of our previous work, where it appears that the use of smaller interconnected overlays has interesting properties concerning scalability and exhaustivity. Section VI presents some examples of application that have been, or could be already developed on top of Synapse, where in Section VII we give conclusions and discuss further work.

II. Related work

Inter-overlay cooperation is a challenge in peer-to-peer networking that has inspired a number of research efforts. These efforts can be classified into the following categories.

A. Cooperation via gateway

The authors in [8] present the model in which dedicated peers are used as gateways for forwarding requests from one overlay to another. However, this model intuitively suffers from the "single point of failure" problem; the gateways are burdened by load and attack. Also, an evaluation of the model hasn't been performed in this work. The authors in [9] propose a protocol using co-located peers and peers whose neighbors belong to other DHTs as virtual gateways to enable cross-DHT searching between various DHT implementations. Although the focus there was on wireless ad-hoc networks, the authors claim that their protocol can be used in wired networks too. Unfortunately, it is unclear how they evaluate their protocol.

B. Cooperation via super overlay

The authors in [10] present a super overlay model, in an effort to enable the interaction of multiple independent overlays. They use a global DHT to organize all the peers across different overlays into a uniform space. The routing between different overlays is performed on the global DHT. The authors in [11] propose another model using a super overlay to enable inter-overlay cooperation. In their model, some chosen peers from more than one overlay are exported and form a Synergy inter-network. When the sender peer and the receiver peer belong to different overlays, they are exported (i.e. join the Synergy network) and, in this way, can use the Synergy path to route to each other.

C. Cooperation via hierarchy

Several works exploit hierarchy to build overlay topologies based on the coexistence of smaller local overlay networks. [12] introduces Canon, that merges the leaf-networks together to form a single overlay. [13] presents the HIERAS overlay, which contains many other overlay networks across different layers inside this overall overlay network. Each sub-overlay contains a subset of the set of all system peers. The authors in [14] propose the model which enables the interconnection of various clusters of peers, by forming an overlay of super peers chosen from these clusters.

The hierarchical overlay approach is certainly the one closest to ours, in terms of goals and implementation. However, we believe that the use of an unstructured routing between overlays can increase the robustness in the case of high churn of gateway nodes.

D. Merging overlay

Some other works attempt to merge several overlays into one overlay. The authors in [8] introduce a protocol that enables one CAN-network to be completely absorbed into another one. However, there is no mathematical analysis or practical evaluation done in this paper. The authors in [15] provide an analysis of the problem of merging two different overlays and introduce an algorithm of merging two ring-based overlays in [16]. Merging overlays require modifying the key space, as well as rearranging keys and data. These tasks are expensive in terms of time and power processing. It also remains impossible to merge heterogeneous overlays.

E. Structured and unstructured overlay cooperation

The authors in [17] introduce a hybrid model, mixing Gnutella and DHT overlays for a file sharing application. The ultra peers in Gnutella form a DHT, and search is performed via conventional flooding techniques of overlay neighbors for normal files, and via DHT queries for rare files.

III. System description

In this section we describe the node structure, inter-overlay routing and node discovery performed in Synapse. As a reference for the operations and messages, we adopt the Key-Based Rotuing API described in [18], since it provides a well-formed general abstraction, independent of specific protocol implementation.

A. General overview and definitions

As we said earlier, the idea behind Synapse is to provide a framework that allows for transparent interconnection and collaboration of heterogeneous overlays. A Synapse node is generally connected to one or more overlay networks (referred to as *Connected Overlays*), and maintains a table of pointers to other Synapse nodes, which can act as gateways to overlays other than the connected ones. These are referred to as *Direct Overlays*. Each overlay network is identified by a unique `networkID`.

Figure 1 depicts an example scenario, where a Synapse node `S1` sees connected overlays `netID1` and `netID5` and direct overlays `netID2`, `netID3`, `netID4`, via gateway nodes `S2`, `S5`, `S8`.

1) General assumptions: The following assumptions have been kept into consideration when designing the protocol:

- All of the overlays expose key based routing capabilities (although an extension to keyword-based searches should be possible);

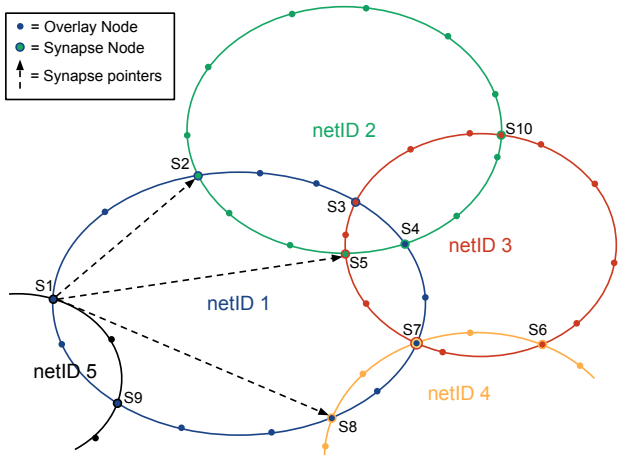

Fig. 1. Synapse overlay overview

- The hash function used by different overlays is not necessarily the same. This involves the exchange of the unhashed key when routing a message outside an onverlay;

2) Synapse definition: To summarize, we can define a *Synapse node* as a node capable of the following operations:

- Processing and routing a message coming from the application layer to the connected overlays;
- Dispatching a message directly to other Synapse nodes, to be routed in overlays others than the connected ones;
- Routing a request coming from other Synapses;
- Discovering new Synapses (Sec. III-D);
- Inviting new Synapse nodes to join connected overlays (Sec. III-F);

Aside from the data structures and messages required to be maintained by each of the connected overlays (finger tables, neighbors list, find_node messages etc.) Synapse introduces new structures to handle the inter-overlay routing:

- a *networkID* per each overlay, to identify it unequivocally;
- a *Direct Overlay Table (DOT)*, that is, a table storing pointers to gateway nodes arranged per networkID;
- a *Message Routing Table (MRT)*, responsible for storing information about ongoing messages (TTL, source nodes, MessageID, targeted overlays...);
- a *Cache table*, used for storing values associated to frequently requested keys, in order to minimize routing for popular items;

Furthermore, Synapse implements the following messages:

- `SYNAPSE_OFFER(netIDList)`, issued by a gateway node to publish the list of overlays it is connected to;
- `SYNAPSE_REQUEST(key, RequestID, message, TTL, strategy, target[networkID], visited[networkID], PubKey)` sent by a synapse node to a gateway in order to route a message outside an overlay;
- `SYNAPSE_RESPONSE(RequestID, netID, responseMessage)` used by a gateway to return the response messages from a foreign overlay;
- `SYNAPSE_INVITE(networkID)` sent to a Synapse node to ”invite” it to join a specific overlay;
- `SYNAPSE_JOIN(networkID)` issued by a Synapse node wishing to join a given overlay;

B. Routing in Synapse

Message routing can follow 3 different mechanisms:

- Synapse nodes can route a message in any of the connected overlays;
- Synapses can also route a message directly to any of the direct overlays by issuing a `SYNAPSE_REQUEST` message to a gateway node in its node table;
- It is also possible to reach *Indirect Overlays*, i.e. overlays not directly reachable, but the networkID of which is known, by issuing a `SYNAPSE_REQUEST` message, with the target networkIDs specified, to a random set of gateway nodes. This engages an unstructured routing through gateway nodes until a node connected to the target overlay is reached;

A `SYNAPSE_REQUEST` message carries within itself several parameters, amongst which the unhashed key for the message, a RequestID (in order to identify if a message has already passed by a gateway), a TTL parameter that defines how many times should a message be routed to subsequent gateways and, if necessary, a list of target networkIDs to which the message should specifically be routed.

The choice of which, and how many overlays to select for message routing constitutes the system’s routing strategy.

C. Routing strategies

A routing strategy consists of a set of rules that regulates the choice of which overlays to route a message to and, to which nodes of said overlays and when to route a message. Routing strategies strongly depend on the application implemented on top of the overlay and the network conditions. Here, we present some examples of possible strategies that can be implemented on top of a Synapse overlay:

1) n-Random Walk: a Synapse node picks *n* random overlays to which to route the request, from all of its connected and direct overlays.

2) n-Flood: a Synapse node picks *n* nodes per *each* direct and connected overlay. The choice of replicating a message on the same overlay comes from the need to overcome network partitioning by routing a request via nodes placed in different locations of the adressing space.

3) Opportunistic routing: a Synapse can dispach a `SYNAPSE_REQUEST` to another Synapse node upon reception of a `SYNAPSE_OFFER`, thus having a much higher chance of routing to an active node.

4) n-Direct routing: a Synapse routes a message directly to a certain overlay only, by picking *n* Synapse nodes connected to said overlay. If no finger to said overlays is present, the message can be routed to random Synapse Nodes by sending

a SYNAPSE_REQUEST message with specified the target networks list.

D. Synapse node discovery strategies

In order to reach direct overlays, a Synapse node that joins the network needs to discover gateway nodes connected to overlays other than his. Depending on the application scenario, there are several mechanisms available:

1) Message embedding: In a collaborative scenario, in which the overlay protocol messages can support additional data, the simplest solution is to embed the list of overlays the node issuing the message is connected to. In this way, each Synapse node forwarding the message in the overlay, can extract this information and update its Direct Overlay Table, in Kademlia-like fashion.

2) Active notifications: Being notified of a transiting message, a Synapse Node can decide to proactively send a SYNAPSE_OFFER message to the source node, containing its overlay list, in order to publish its presence. This is an effective technique in non-collaborative scenarios, in which a message source is known (e.g. iterative or semi-recursive routing protocols).

3) Peer exchange: For those scenarios in which embedding is not possible, and a message source is not known (due to a fully recursive routing algorithm), a Synapse Node can still discover other Synapses via an iterative peer exchange mechanism. This, however, requires an initial Synapse bootstrap node to be contacted, in order to perform the first discovery.

4) Aggressive discovery: Aside from the above strategies, which are generic and suitable for any overlay protocol, other strategies can be put in place in a collaborative scenario, to exploit the specificity of a certain protocol (e.g. a source node list, leaf tables, neightbor cache...).

E. Synapse node structure

As shown in Figure 2, a Synapse Node instance is made up of several components:

- The *Synapse Application Adapter* acts as an interface to and from the Application layer. It serves the purpose of decoupling the applicative part from the multi-overlay logic behind by exposing an API agnostic of the undrlying structure, processing complex queries, and to generating the appropriate messages for the Synapse Controller;
- The *Synapse Controller* is reponsible for orchestrating multiple requests, routing messages according to the appropriate strategy, and collecting and grouping results coming from different overlays. It also takes care of the Synapse overlay maintenance, by performing the discovery of new Synapse nodes, checking their state via ping messages and dispatching join invitations to modify the overlay topology (see Sec.III-F). It maintains and relies on the *Direct Overlay Table (DOT)* to store pointers to gateway nodes and the *Message Routing Table (MRT)*, to keep track of ongoing routings. Furthermore, the *Cache Table* can store values retrieved recently, in order to serve them immediately should a new request for the same key arrive. The Synapse Controller contacts its direct overlays by sending ROUTE messages to the overlay sub-modules. Each overlay sub-module, on the other hand, notifies the Synapse Controller via a NOTIFY message every time an overlay message is forwarded by them, or an RCP message is received, in order to check for new gateway nodes by examining whether or not a networkID list is present in the message header, or to announce its own presence via a SYNAPSE_OFFER message.

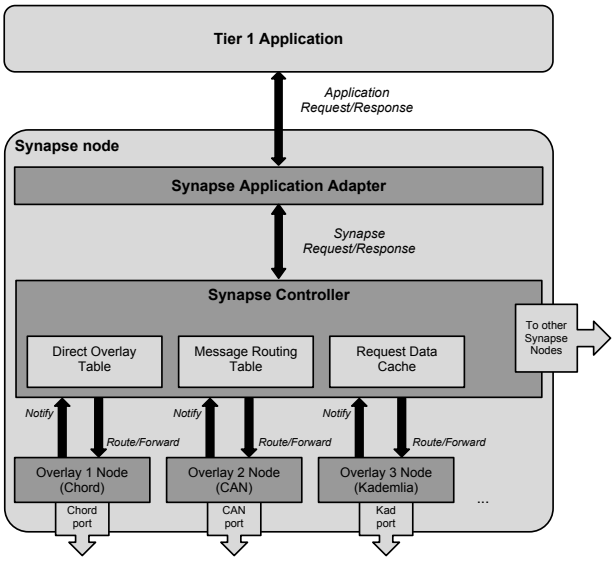

Fig. 2. Structure of a Synapse Node

F. Self-organization via "social networking" primitives

In addition to Synapse messages, we propose a set of primitives to implement overlay self-organization mechanisms. Through the issue of a SYNAPSE_INVITE message, a Synapse node can ask other Synapses to join one or more overlays, in order to increase the overlay capacity, QoS or external connectivity.

In the same way, a Synapse node can propose itself to be a member of an overlay, with a SYNAPSE_JOIN message addressed either to another Synapse node which is already a member of the target overlay, or to an authentication server.

Social based primitives are particularly interesting if considered under the perspective of being able to have an overlay "grow" or "shrink" around application data, such as, for example, the social graph in online social networks. They can also be exploited to regulate the connectivity of an overlay towards the rest of the system, by increasing the number of gateways to said overlays, providing a flexible mechanism to implement QoS and failure avoidance in a system.

G. Routing example

We hereby present an example of routing in a Synapse network, using a Random Walk strategy with opportunis-

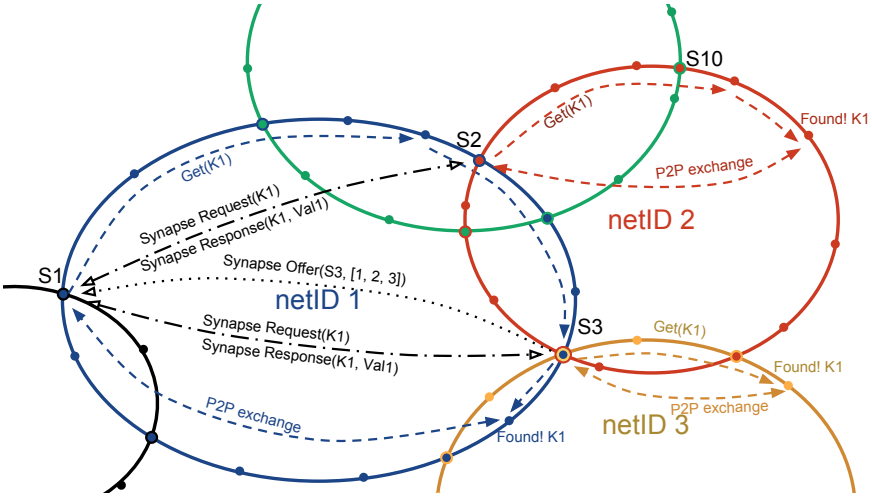

Fig. 3. Routing in Synapse

tic routing enabled. Figure 3 shows the message exchange between nodes. For the example, we consider a DHT-like application where chunks of data, associated with keys, can be spread and replicated into multiple overlays. In our case, node *S1* wants to retrieve the data associated with key `K1`. The following operations are involved:

1) The Application Layer on node *S1* sends a `GET(K1)` to the Synapse Controller via internal APIs.
2) The Synapse Application Adapter, aware of the organization of data in the system, translates it into a `MULTI_GET(K1, strategy=RANDOM.1.1)` message to the Synapse Controller, specifying the key to be retrieved and the strategy to adopt.
3) The Synapse Controller, according to the requested 1-Random-Walk strategy, picks 1 random overlay (*netID1*) from the connected overlay list and 1 random node (*S2*) from the Direct Overlay Table.
4) It routes directly a `GET(Hash(K1))` message in *netID1*, hashing the key with the overlay's hash function.
5) In parallel, it generates a new *RequestID* and sends a `SYNAPSE_REQUEST(K1, RequestID, TTL=1, strategy=RANDOM.1.1, visited=[netID1], S1PubKey)` to *S2*.
6) Upon reception of said message, *S2* picks 1 random connected overlay (*netID2*), avoiding *netID1*, to reroute the request and decreases the TTL value. Since now TTL=0, the request is not routed any further to other gateway nodes.
7) During the routing in *netID1*, the message is forwarded by another gateway node (*S3*) connected to netID1. *S3*, first updates its DOT with node *S1* and its netID list embedded in the message, then notifies *S1* of its presence by sending it a `SYNAPSE_OFFER(myList=[netID1, netID2, netID3], S3PubKey)`.
8) *S1*, upon reception of the SYNAPSE_OFFER, first updates its DOT with *S3*, then replies with a `SYNAPSE_REQUEST(K1, ReqID, TTL=1 RANDOM.1.1, [O1, O2], S1PubKey)`.
9) *S3*, receiving the `SYNAPSE_REQUEST`, picks overlay *netID3*. and routes a `GET(Hash(K1))`. Since *TTL* is now 0, the request is not routed to any other gateway.
10) Eventually, the request in *netID1* will reach its destination node, and a response will be sent back to *S1*.
11) The same message will reach its destination in *netID2* and *netID3* and responses will be routed back to nodes *S2* and *S3*, who will then send the message response `RESP1` (containing the value associated to `K1`) back to *S1* via a `SYNAPSE_RESPONSE(ReqID, RESP1, O2)` message, encrypted with *S1*'s public key.
12) Once all the responses have been gathered they are sent up to the Application Adapter. Depending on the application it has several possibilities, for example sending back the whole dataset, randomly select one of the retrieved values, pick the most recent or perform a majority selection.

From this example there appear different interesting properties of the protocol:

- By routing recursively, node *S1* is not exposed in overlays where it is not connected to.
- The key is sent out un-hashed only in the `SYNAPSE_REQUEST` messages, which are encripted via a public key mechanism.
- Routing in direct overlays takes only 2 more hops more than if *S1* was connected to them, 1 hop for the `SYNAPSE_REQUEST` and 1 for the `SYNAPSE_RESPONSE` to travel back.
- During the routing in *netID1*, *S1* came to discover a new direct overlay, `netID3`, which then becomes a direct overlay accessible by contacting *S3*.

IV. Protocol implementation in OverSim simulator

To precisely capture the behaviour of traditional metrics of overlay networks under controlled conditions, we implemented our Synapse protocol in the OverSim Overlay Simulator [19]. OverSim is an overlay network simulator implemented on top of the Omnet++ framework [20]. Its choice was dictated by the following reasons:

- It provided a whole set of overlay protocols already implemented and tested, such as Chord, Kademlia, Pastry, Koorde etc., in both the iterative and recursive form.
- Being based on Omnet++, it brought with itself an excellent configuration framework, as well as all the logic behind it.
- It already captures relevant overlay network statistics, such as exchanged messages, dropped packets, latencies, and highlighting relevant information.
- Thanks to the Omnet++ framework, it is possible to run a simulation in a cluster, using the MPI framework.
- It allows the use of different libraries to simulate the underlay layer, includes a module to perform actual deployment of simulation code and the exchange of messages on a real network.
- It provides classes and methods suitable for the implementation of new overlay protocols and applications on top, with the minimum amount of code, exposing a clear API derived from [18].

However, the implementation of Synapse on top of OverSim presented several challenges due to the internal architecture, which is designed to support either one overlay, or multiple overlays of the same type. Without going too much into technical detail, we briefly describe how such challenges have been overcome, as they could be of interest to anyone else involved in the development of heterogeneous overlays inside OverSim.

A. Extending OverSim's overlay host

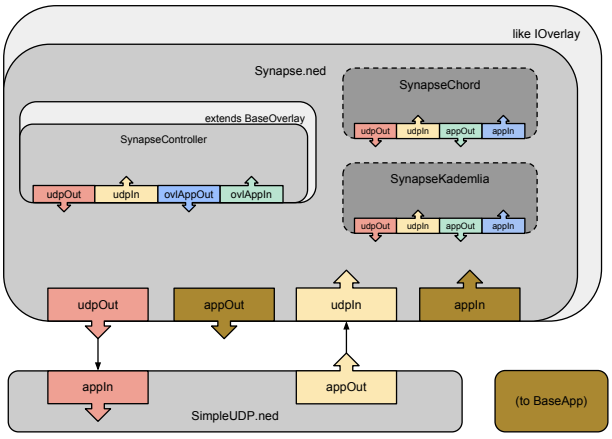

Fig. 4. Synapse OverSim modules diagram

Figure 4 shows the Synapse Controller module diagram in Oversim. The same colors for 2 gates indicate that the gates are connected. The controller has been implemented as a `BaseOverlay` derived class, so that Tier-n modules could see it as the only overlay module and, thus, be decoupled by the multi-overlay routing. However, the Synapse Controller implements a double behaviour, with relation to the OverSim model, acting also as a Tier-1 application connected to the overlays' submodules (`SynapseChord`, `SynapseKademlia` etc.) via the `ovlAppin/out` gates. In this way, it is able to control the overlays by using the `CommonAPI` messages provided by OverSim, without any changes to the overlay submodules logic.

To work around the static architecture mentioned before, the overlay submodules have to be instantiated at runtime during the Synapse Controller `INIT` phase. In addition to allowing an extremely granular configuration of the overlays (the initial overlay interconnection can be setup for each individual node), manual instantiation of overlay modules becomes necessary when implementing the social networking logic, since a Synapse node might have to join a new overlay at runtime.

B. Extending OverSim's overlay modules

The goal for this implementation was to leave OverSim overlay modules code untouched, in order not to break compatibility or to generate unwanted behaviors. The only additional operation to be implemented in each of the overlays was to send a notification message (`KBRNotify`) to the Synapse Controller each time a message was routed or an RCP call was received. Through the use of template metaprogramming, we managed to implement all of the required logic in a wrapper class, `SynapseOverlayWrapper` that could inherit any of the overlay classes, which are passed as parameters of the template.

Here is the class definition for the SynapseOverlayWrapper:

```
template
<class BaseOverlayType=BaseOverlay>
class SynapseOverlayWrapper :
  public BaseOverlayType
```

This allows us to create extended classes by a simple inheritance mechanism. The `SynapseChord` class is defined as follows:

```
class SynapseChord :
  public SynapseOverlayWrapper<Chord>
```

Thanks to this parametrized inheritance, the SynapseOverlayWrapper can access any attribute or protected member of `BaseOverlay`, which every overlay module inherits, while leaving the specific implementation untouched.

The simulator code is open source and it is available at [21].

V. Expected results

Synapse is still a work in progress and extensive simulations are currently ongoing to thoroughly test the scalability and

exhaustivity in different network scenarios and with different routing strategies, illustrating possible application uses, and we are confident in the capabilities of our architecture.

To support that, we briefly present some results of an early work [22], where an initial version of Synapse was implemented and tested. This simulations show the performaces of what is now called the opportunistic strategy to perform inter-overlay routing, without the use of the Direct Overlay Table. It can be considered as a worst-case scenario, in which peers do not discover gateway nodes, nor perform any peer exchange.

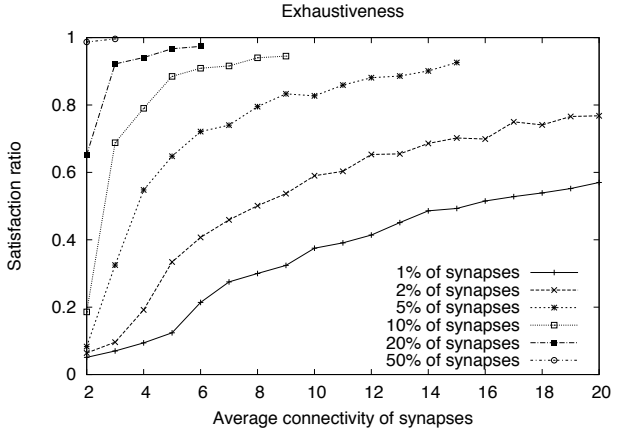

Fig. 5. Lookup exhaustiveness with opportunistic routing

Figure 5 shows the routing exhaustiveness, while varying the granularity of the network, the percentage of gateway nodes per overlay and the average interconnection degree per each gateway.

The figure shows that a low synapse degree (2) is enough to achieve quasi-exhaustiveness, which is a somewhat unexpected and seemingly good result, given in mind that we are trying to limit the cost of each gateway node to maintain a connection to several overlays. Furthermore, the granularity does not significantly influence exhaustiveness when the number and connectivity of the synapses are fixed.

Figure 6, on the other hand, shows the average latency (measured in number of hops) observed when retrieving a key in a random overlay. An interesting point to notice is that the number of hops remains logarithmic when changing from a Chord network into a Synapse network (the number of nodes is 10000, the latency never exceeds 14). Other simulations, are in accordance with this observation. We adress the reader to [22] for a more detailed discussion over the results.

VI. Application Scenarios

We herein present some examples of distributed applications that could sit on top of a multiple overlay structure provided by the Synapse protocol, as well as some interesting aspects and possibilities which emerge when thinking about possible applications. As one can imagine, most of the examples exploit the increase locality offered by a federated network, be it network, geographical, social or semantic locality.

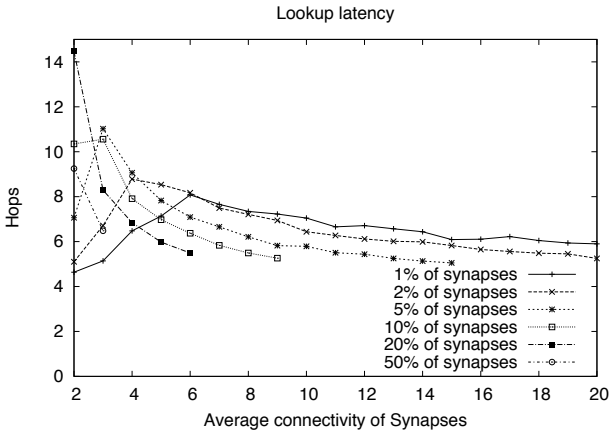

Fig. 6. Lookup latency with opportunistic routing

A. P2P Online Social Networks

In [23] the authors describe a partitioning algorithm for the Cassandra DHT which optimizes the key in such a way that data related to users close in the social graph has a greater chance of residing in the same Cassandra node.

Such an algorithm could be easily adapted to a multi-overlay scenario based on Synapse, allowing users to share their data in one or more DHTs, whose nodes are 1st or 2nd degree links in their social graph, allowing for a scalable implementation of a real P2P OSN.

B. Community-centric publish/subscribe

In [24] we described a proof of concept of a car sharing application based on an early design of the Synapse protocol. Such an application would allow members of a social community (e.g. school, company etc.) to share, through the use of a Distributed Hash Table, service offerings, in this case car rides. Through the interconnection of different DHTs, it was possible to increase the chance of matching a demand with its correspondent offering, by extending the queries to communities based on geographic proximity, which are more prone to having the same travel patterns.

C. Security and anonimity

Network federation can allow for the creation of trusted overlays, in which data is stored only by a set of trusted peers. Data replication across multiple overlays can lead to more efficient strategies against data pollution, where multiple queries for the same key can be issued in several overlays, only to choose the value returned by the majority of the overlays.

Moreover, messages traveling in one overlay can hardly be traced back, since a request could have always come on behalf of a foreign node.

Potentially, in a fully recursive protocol setup, the only message carrying the unhashed key and the source node is the `SYNAPSE_REQUEST`, which, being a point-to-point message, can be easily encrypted via a public key mechanism.

D. Database interoperability

In [25], we present another proof of concept that exploits the Synapse protocol to allow the interconnection of distributed digital catalogs of cultural heritage documents. In this example, while the document was still under the control of the owner organization, the document metadata could be shared in several DHTs, one for each of the organizations, allowing users to access the global data via extended queries.

VII. Conclusions and Further Work

In this paper, we have presented a novel architecture, the purpose of which is to enable the design of distributed applications based on multiple interconnected overlays, as well as to facilitate easier interconnection of existing overlays.

As we have just begun scratching the surface of all the possibilities offered by such an approach, our further work includes a mathematical modeling of the system, and an extensive testing of all the routing strategies in order to be able to accurately quantify messaging overhead, resilence to churn and data consistency. Furthermore, we need to be able to define a mechanism which would guarantee a minimum level of interconnection between different overlays, i.e. to assure a constant presence of only a minimal number of gateway nodes within the overlays.

A study about the integration of Synapse with keywork-based unstructured overlays is under way, and the possibility of compiling keyword-based queries into key-based ones is being carefully considered.

Currently, a java-based client exploiting well-tested overlay protocol libraries, as well as simulation code on top of the OverSim simulator [19] have been developed and are under test.

References

[1] R. Jimenez, F. Osmani, and B. Knutsson, "Connectivity properties of mainline bittorrent dht nodes," in *Proceedings of IEEE P2P '09.*, 2009.

[2] ——, "Sub-Second lookups on a Large-Scale Kademlia-Based overlay," in *Proceedings of 11th IEEE International Conference on Peer-to-Peer Computing 2011*, 2011.

[3] P. Cudré-Mauroux, S. Agarwal, and K. Aberer, "GridVine: An infrastructure for peer information management," *IEEE Internet Computing*, vol. 11, no. 5, 2007.

[4] M. Varvello, C. Diot, and E. Biersack, "A walkable kademlia network for virtual worlds," in *Proceedings of IPTPS*, 2009.

[5] G. Urdaneta, G. Pierre, and M. V. Steen, "A survey of dht security techniques," *ACM Comput. Surv.*, vol. 43, 2011.

[6] Bittorrent website. http://www.bittorrent.com. [Online]. Available: http://www.bittorrent.com

[7] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric," in *Proc. of 1st International Workshop on Peer-to-peer Systems*, 2002.

[8] L. Cheng, R. Ocampo, K. Jean, A. Galis, C. Simon, R. Szabó, P. Kersch, and R. Giaffreda, "Towards distributed hash tables (de)composition in ambient networks," in *Proceedings of DSOM*, 2006.

[9] L. Cheng, "Bridging distributed hash tables in wireless ad-hoc networks," in *Proceedings of GLOBECOM*, 2007, pp. 5159–5163.

[10] G. Tan and S. A. Jarvis, "Inter-overlay cooperation in high-bandwidth overlay multicast," in *Proceedings of ICPP*, 2006, pp. 417–424.

[11] M. Kwon and S. Fahmy, "Synergy: an overlay internetworking architecture," in *Proceedings of ICCCN*, 2005.

[12] P. Ganesan, P. K. Gummadi, and H. Garcia-Molina, "Canon in g major: Designing dhts with hierarchical structure," in *Proceedings of ICDCS*, 2004.

[13] Z. Xu, R. Min, and Y. Hu, "Hieras: A dht based hierarchical p2p routing algorithm," in *Proceedings of ICPP*, 2003.

[14] L. Garcés-Erice, E. W. Biersack, P. Felber, K. W. Ross, and G. Urvoy-Keller, "Hierarchical peer-to-peer systems," in *Proceedings of Euro-Par*, 2003.

[15] A. Datta and K. Aberer, "The challenges of merging two similar structured overlays: A tale of two networks," in *Proceedings of IW-SOS/EuroNGI*, 2006.

[16] T. M. Shafaat, A. Ghodsi, and S. Haridi, "Dealing with network partitions in structured overlay networks," *Peer-to-Peer Networking and Applications*, vol. 2, no. 4, 2009.

[17] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein, "The case for a hybrid p2p search infrastructure," in *Proceedings of IPTPS*, 2004.

[18] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a common api for structured peer-to-peer overlays," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, 2003.

[19] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, 2007.

[20] Omnet++ network simulator. http://www.omnetpp.org. [Online]. Available: http://www.omnetpp.org

[21] Synpase oversim implementation webpage. http://www-sop.inria.fr/teams/lognet/synapseV2.0/. [Online]. Available: http://www-sop.inria.fr/teams/lognet/synapseV2.0/

[22] L. Liquori, C. Tedeschi, L. Vanni, F. Bongiovanni, V. Ciancaglini, and B. Marinkovic, "Synapse: A scalable protocol for interconnecting heterogeneous overlay networks," in *Proceedings of Networking*, 2010.

[23] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, "The little engine(s) that could: scaling online social networks," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, 2010.

[24] V. Ciancaglini, L. Liquori, and L. Vanni, "Carpal: Interconnecting overlay networks for a community-driven shared mobility," in *Proceedings of Trustworthly Global Computing Conference*, 2010.

[25] B. Marinković, L. Liquori, V. Ciancaglini, and Z. Ognjanović, "A distributed catalog for digitized cultural heritage," in *Proceedings of ICT Innovations 2010*, 2011.