Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks*

Luigi Liquori[1]**, Cédric Tedeschi[2], Laurent Vanni[1],
Francesco Bongiovanni[1], Vincenzo Ciancaglini[1], and Bojan Marinković[3]

[1] Institut National de Recherche en Informatique et Automatique, France
Email: firstName.lastName@sophia.inria.fr
[2] Université de Rennes I/INRIA, France
Email: Cedric.Tedeschi@inria.fr
[3] Mathematical Institute of the Serbian Academy of Sciences and Arts, Serbia
Email: bojanm@turing.mi.sanu.ac.rs

Abstract. This paper presents Synapse, a scalable protocol for information retrieval over the inter-connection of heterogeneous overlay networks. Applications on top of Synapse see those intra-overlay networks as a unique inter-overlay network. Scalability in Synapse 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 *neural synapses* and connected to several overlay networks, allow a larger search area and provide alternative routing. Synapse can either work with "open" overlays adapting their protocol to synapse interconnection requirements, or with "closed" overlays that will not accept any change to their protocol.
Results from simulation and experiments show that Synapse is scalable, with a communication and state overhead scaling similarly as the networks interconnected. Thanks to alternate routing paths, Synapse also gives a practical solution to network partitions.
We precisely capture the behavior of traditional metrics of overlay networks within Synapse and present results from simulations as well as some actual experiments of a client prototype on the Grid'5000 platform. The prototype developed implements the Synapse protocol in the particular case of the interconnection of many Chord overlay networks.

Keywords. Peer-to-peer, overlay networks, information retrieval.

1 Introduction

Context. The interconnection of overlay networks has recently been identified model showing great promise when it comes to dealing with the issues of the Internet of today, such as scalability, resource discovery, failure recovery or routing efficiency and, in particular, in the context of information retrieval. Several recent research efforts have focused on the design of mechanisms for building bridges between heterogeneous overlay networks for the purpose of improving cooperation between networks which have different routing mechanisms, logical topologies and maintenance policies. However, more comprehensive studies of such interconnections for information retrieval and both quantitative and experimental studies of its key

* Supported by AEOLUS FP6-IST-15964-FET Proactive and DEUKS JEP-41099 TEMPUS.
** Corresp. author. Thanks to anonymous referees and Ernst Biersack for the precious discussions.

metrics, such as satisfaction rate or routing length, are still missing. During the last decade, different overlay networks were specifically designed to answer well-defined needs such as content distribution through unstructured overlay networks (Kazaa) or through structured networks, mainly utilizing concepts such as Distributed Hash Tables [14, 15, 17] and publish/subscribe systems [2, 12].

An overview of the current problem. Many disparate overlay networks may not only simultaneously co-exist in the Internet but also compete for the same resources on shared nodes and underlying network links. One of the problems of the overlay networking area is how heterogeneous overlay networks may *interact* and *co-operate* with each other. Overlay networks are heterogeneous and basically unable to cooperate each other in an effortless way, without merging, an operation which is very costly since it not scalable and not suitable in many cases for security reasons. However, in many situations, distinct overlay networks could take advantage of cooperating for many purposes: collective performance enhancement, larger shared information, better resistance to loss of connectivity (network partitions), improved routing performance in terms of delay, throughput and packets loss, by, for instance, cooperative forwarding of flows.

As a basic example, let us consider two distant databases. One node of the first database stores one $(key, value)$ pair, which is then requested by a node of the second database. Without network cooperation those two nodes will never communicate together. In another example, we have an overlay network in which a number of nodes got isolated by an overlay network failure, leading to a partition: if some or all of those nodes are reachable via an alternate overlay network, than the partition "could" be recovered via an alternate routing.

In the context of large scale information retrieval, several overlays may want to offer an aggregation of their information/data to their potentially common users without giving up control over it. Imagine two companies wishing to share or aggregate information contained in their distributed databases, while, obviously, keeping their proprietary routing and their exclusive right to update it. Finally, in terms of fault-tolerance, cooperation can increase the availability of the system, if one overlay becomes unavailable, the global network will only undergo partial failure as other distinct resources will be usable.

We consider the tradeoff of having one *vs.* many overlays as a conflict without a cause: having a single global overlay has many obvious advantages and is *de facto* the most natural solution, but it appears unrealistic in the actual setting. In one optimistic case, different overlays are suitable for collaboration by opening their proprietary protocols in order to build an open standard; in many other pessimistic cases, this opening is simply unrealistic for many different reasons (backward compatibility, security, commercial, practical, etc.). With all this said, studying protocols to interconnect collaborative (or competitive) overlay networks appears to be quite an interesting and intriguing research vein.

Contribution. The main contributions of this paper are the introduction of *Synapse*, a scalable protocol for information retrieval over the inter-connection of heterogeneous overlay networks, and the presentation of the results obtained from various simulations and experiments with this protocol. The protocol itself is based on co-located nodes, also called *synapses*, serving as low-cost natural candidates for inter-overlay bridges. In principle, every regular node can become a synapse. 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 synapse 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 of the other overlay networks it belongs to. This implies that a Time-To-Live value is provided, and already processed queries can be detected, so as to avoid infinite looping in the network, as in unstructured peer-to-peer systems.

In case of concurrent overlay networks, inter-overlay routing becomes harder, as intra-overlays are provided as 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, since we would like to route queries along closed overlays and prevent routing loops.

Our experiments and simulations show that a small number of well-connected synapses is sufficient in order to achieve almost exhaustive searches in a "synapsed" network of structured overlay networks. We believe that Synapse can give an answer to circumventing network partitions; the key points being that: (i) several logical links for one node lead to as many alternative physical routes through these overlays, and (ii) a synapse can retrieve keys from overlays that it does not even know simply by forwarding its query to another synapse that, in turn, is better connected. Those features are achieved in Synapse at the cost of some additional data structures and in an orthogonal way to ordinary techniques of caching and replication. Moreover, being a synapse can allow for the retrieval of extra information from many other overlays even if the node isn't directly connected with them. We summarize our contributions as follows: (i) the introduction of *Synapse*, a generic protocol, which is suitable for interconnecting heterogeneous overlay networks without relying on merging in presence of open/collaborative or closed/competitive networks; (ii) extensive simulations in the case of the interconnection of structured overlay networks, in order to capture the real behavior of such platforms in the context of information retrieval, and identify their main advantages and drawbacks; (iii) the deployment of a lightweight prototype of Synapse, called `JSynapse` on the Grid'5000 platform[4] along with some real deployments showing the viability of such an approach while validating the software itself; (iv) finally, on the basis of the previous item, the description and deployment on the Grid'5000 platform of a open source prototype, called `open-Synapse`, based on the `open-Chord`[4] implementation of Chord, interconnecting an arbitrary number of Chord networks. The final goal is to grasp the complete potential that co-located nodes have to offer, and to deepen the study of overlay networks' inter-connection using these types of nodes.

Outline. The remainder of the paper is organized as follows: In Section 2, we introduce our Synapse protocol at work for open/collaborative overlays (*white box*) viz. closed/competitive (*black box*) overlays, and provide several examples. In Section 3, we present the results of our simulations of the Synapse protocol to capture the behavior of key metrics traditionally used to measure the efficiency of information retrieval. In Section 4, we describe a deployment of a client prototype[5] over the Grid'5000 platform. In Section 5, we summarize the mechanisms proposed in the literature related to the cooperation of overlay networks. Finally, in Section 6, we present our conclusions and

[4] `http://www.grid5000.fr` and `http://open-chord.sourceforge.net`.

[5] Code and web appendix at `http://www-sop.inria.fr/teams/lognet/synapse`.

discuss ideas for further work. Appendix shows pseudocode: due to lack of space, the protocol is presented in details in a separate web appendix[5].

2 The Synapse Protocol

Architecture and assumptions. We now present our generic *meta*-protocol for information distribution and retrieval over an interconnection of heterogeneous overlay networks. Information is a set of basic `(key,value)` pairs, as commonly encountered in protocols for information retrieval. The protocol specifies how to insert information (`PUT`), how to retrieve it through a key (`GET`), how to invite nodes in a given overlay (`INVITE`), and how to join a given overlay (`JOIN`) over a heterogeneous collection of overlay networks linked by co-located nodes. These co-located nodes represent a simple way to aggregate the resources of distinct overlays. We assume each overlay to have its own inner routing algorithm, called by the Synapse protocol to route requests inside each overlay. We assume no knowledge of the logical topology of all the involved overlay networks connected by Synapse. To ensure the usual properties of the underlying network, we assume that communication is both symmetric and transitive. Synapse simply ignores how routing takes place inside the overlays, Synapse only offers a mechanism to route from one overlay to another in a simple, scalable and efficient way.

The inter-overlay network, induced by the Synapse protocol, can be considered as an aggregation of heterogeneous sub-overlay networks (referred to as *intra*-overlay networks henceforth). Each intra-overlay consists of one instance of, *e.g.*, Chord or any structured, unstructured, or hybrid overlay. We recall that an overlay network for information retrieval consists of a set of nodes on which the information on some resources is distributed. Each intra-overlay has its own key/value distribution and retrieval policy, logical topology, search complexity, routing and fault-tolerance mechanisms, so on and so forth. The Synapse protocol can be summarized by the following points:

- *Synapses:* the interconnection of intra-overlay networks is achieved by co-located nodes taking part in several of these intra-overlays, called synapses. Each peer will act in accordance with the policies of each of its intra-overlays, but will have the extra-role of forwarding the requests to some other intra-overlay it belongs to. As stated in the introduction, every node can become a synapse.
- *Peer's name:* every peer comes with a proper logical name in each intra-overlay; in particular, synapses have as many logical names as the number of networks they belong to.
- *Keys mapping in peers:* each peer is responsible for a set of resources (`key,value`) it hosts. Since every intra-overlay has different policies for key distribution, we could say that the inter-overlay induced by Synapse also inherits homogeneous distribution among the intra- and inter-networks. As for peers, every key comes with a proper logical name peculiar to each intra-overlay.
- *Set of resources assigned to set of nodes:* all overlay protocols for information retrieval share the invariant of having a set of peers responsible for a specific set of resources. This invariant allows for routing under structured, unstructured and hybrid networks because, by construction, intra-routing is the one always responsible for its correctness, since Synapse only cares about the overlay's interconnection.

- *Network in dependency and message translation:* intra-network protocols are different by construction: as such, when a message leaves a particular network and enters another network, the first network loses control of the route of that message inside the second one.
- *Topology, exhaustiveness, complexity and scalability:* by construction, the inter-overlay network induced by the Synapse protocol belongs to the category of unstructured overlay networks, with a routing that is not exhaustive, even if Synapse can connect only overlays that guarantee exhaustivity. The same goes for the routing complexity that can be upper-bounded only in the presence of precise and strong hypotheses about the type of intra-overlay networks. The same goes for scalability: a Synapse inter-network is scalable if all of the intra-networks are scalable.
- *Loopy routing avoidance:* to avoid lookup cycles when doing inter-routing, each peer maintains a list of tags of already processed requests, in order to discard previously seen queries, and a TTL value, which is decreased at each hop. These two features prevent the system from generating loops and useless queries, thus reducing the global number of messages in the Synapse inter-network.
- *Replications and Robustness:* to increase robustness and availability, a key can be stored on more than one peer. We introduce a Maximum-Replication-Rate (MRR) value which is decreased each time a `PUT` message touches a synapse, thus replicating the resource in more than one intra-overlay. This action acts as a special TTL denoting how many overlays can traverse a `PUT` message.
- *Social primitives:* each peer implements autonomously a `good_deal?` policy. This is a social-based primitive aimed at making some important choices that may strongly influence the performance and robustness of the Synapse routing. In particular, such a primitive is intended to make easier the choice of whether or not to join another intra-overlay (hence to become a synapse), invite or accept a peer to one of the overlays (hence invite a peer to become a synapse), or even create a new network from scratch. There is no best good deal strategy: for example, if one network wants to increase connectivity with other overlays, it can suggest to all peers to invite and join all interesting/interested peers: this can be especially useful in case of high churning of the intra-network in order to increase alternative routing-paths through the neighboring intra-networks.

"White box" *vs.* **"black box" synapse protocol.** As stated in the introduction, one important issue in interconnecting overlay networks is the ability of one overlay to potentially modify its protocol instead of only accepting that co-located nodes will route packets without any change in the protocol itself. This is a concrete backward compatibility issue, since many overlays already exist, and it is hard to change them at this point for many reasons (security, commercial, technological ...).

As such, we have developed two variants of the synapse protocol; the first *white box* variant, is suitable for interconnecting overlays whose standards are open and collaborative, meaning that the protocol and the software client can be modified accordingly. The second, *black box* variant, is suitable for interconnecting overlays that, for different reasons, are not collaborative at all, in the sense that they only route packets according to their proprietary and immutable protocol. The white box allows for the adding of extra parameters to the current inter-overlay we are connecting, while the black box deals with those extra parameters by means of a *synapse control network*, *i.e.* a distributed overlay that stores all the synapse parameters that cannot be carried on by the overlay we are traversing.

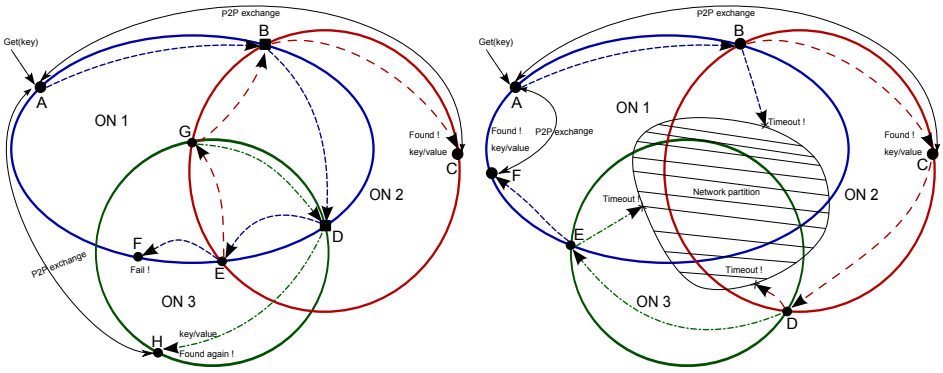

Fig. 1. Routing across different overlays and dealing with a network partition

White box synapse. The white box presented here is capable of connecting heterogeneous network topologies, given the assumption that every node is aware of the additions made to existing overlay protocols. The new parameters used to handle the game over strategy and replication need to be embedded into the existing protocols, and so does the unhashed key in order to be rehashed when a synapse is met. One important requirement of the Synapse white box protocol with respect to other protocols using hash functions is that the keys and nodes' addresses circulate *unhashed* from hop to hop. Hash functions have no inverse: once a sought key is hashed, it is impossible to retrieve its initial value, and thus impossible to forward to another overlay having a different hash function, since hash functions may vary (in implementations and key size) from overlay to overlay. Both the hashed and the *clear* key data can be carried within the message, or a fast hash computation can be performed at each step. Standard cryptographic protocols can be used in case of strong confidentiality requirements, without affecting the scalability of the Synapse protocol itself.

Black box synapse. Interconnecting existing overlays made of "blind" peers, who are not aware of any additional parameters, seems to be a natural Synapse evolution and it constitutes a problem worth investigating. The assumption is that an overlay can be populated by blind peers (*e.g.* nodes previously in place) and synapses at the same time. Both interact in the same way in the overlay and exchange the same messages; moreover, those synapses can be members of several overlays independently (thus being able to replicate a request from one overlay to another) and can communicate with each other exclusively through a dedicated *Control Network*. The Control Network is basically a set of DHTs allowing each node to share routing information with other synapses without being aware of the routing of the undergoing message. So far the DHTs implemented are the following: (i) a Key table, responsible for storing unhashed keys circulating in the underlying overlays. Every synapse accessing this table can easily retrieve the key in clear way using only the information it is aware of; (ii) a Replication table, in which the number of times the key should be replicated across all of the the overlays is stored; (iii) a Cache table, used to implement the replication of `GET` requests, cache multiple responses and control the flooding of foreign networks. Due to the obvious lack of space, the white and black box models are treated in the web appendix.

Example 1. Routing across different intra-overlays. Figure 1 shows how a value present in one overlay can be retrieved from a `GET` launched by another overlay. Peer A in the overlay ON1 receives a `GET(key)` message: the routing goes until synapse B, which triggers a second intra-overlay routing in ON2. The two routings proceed in parallel, and, in particular, the routing in ON2 terminates successfully with a peer-to-peer interaction between peer A and peer C, responsible for the resource. Routing continues on ON1 until synapse D, which triggers a third intra-overlay routing in ON3. The routing proceeds in parallel, and, in particular, routing in ON3 terminates successfully with a second peer-to-peer interaction between A and H, while routing in ON1 proceeds to a failure on peer F via synapse E. Synapse E launches a fourth intra-overlay routing in ON2 that proceeds to a failure on node B (game over strategy) via synapse G. Finally, G launches a fifth intra-overlay routing on ON3, terminating with a failure on D (game over strategy again). Peers playing the game over strategy are depicted as squares.
Example 2. Dealing with network partition. Figure 1 also shows how intra-overlays take advantage of joining each other in order to recover situations where network partitioning occurs (because of the partial failure of nodes or the high churn of peers). Since network partitions affect routing performance and produce routing failures, the possibility of retrieving a value in a failed intra-overlay routing is higher, thanks to alternate inter-overlay paths. More precisely, the figure shows how a value stored in peer E of the overlay ON1 can be retrieved in presence of a generic network partition by routing via ON2 and ON3 through synapses B,C,D, and E. Please refer to the appendix for a description of the protocol pseudocode, in both the white and the black box model.

3 The Simulations

The purpose of the simulations is a better understanding of the behavior of platforms interconnecting structured overlay networks through the Synapse approach. We focus on the key metrics traditionally considered in the distributed information retrieval process, such as exhaustiveness (the extent of existing objects effectively retrieved by the protocol), latency (number of hops required to reach the requested object) and the amount of communications produced (number of messages generated for one request). We want to highlight the behavior of these metrics while varying the topology (the number of synapses and their connectivity, TTL, the number of intra-overlays ...).
Settings. Our simulations have been conducted using Python scripts, and using the *white box* protocol, capturing the essence of the Synapse approach. The topology of the overlay simulated is a set of Chord sub-networks interconnected by some synapses. Information is a set of `(key,value)` pairs. Each pair is unique and exists once and only once in the network. We study the unstructured interconnection of structured networks. We rely on discrete-time simulation: queries are launched on the first discrete time step, initiating a set of messages in the network, and each message sent at the current step will be received by its destination (next routing hop) at the next time step.
Synapses. Our first set of simulations had the intent of studying how the previously mentioned metrics vary while we add synapses or increase the degree of existing ones (the number of intra-overlays a co-located node belongs to). In this first set of simulations, the topology was built beforehand. The number of nodes was fixed to 10000, uniformly distributed amongst 20 overlays (*i.e.*, approximately 500 nodes within each Chord). Queries are always triggered by one random node, the key sought by a

query is also picked uniformly at random from the set of keys stored by the network. A query is said to be *satisfied* if the pair corresponding to the key has been successfully retrieved.

We first studied search latency, *i.e.* the number of hops needed to obtain the first successful response. As illustrated in Figure 2, the first point to notice is that the number of hops remains logarithmic when changing a Chord network into a Synapse network (if the number of nodes is 10000, then the latency never exceeds 14). Other experiments conducted by increasing the number of nodes confirm this. More precisely, (top left) highlights the following behavior: (i) when the network contains only a few synapses, the latency first increases with the degree of synapses: only a few *close* keys are retrieved (keys available in the network of the node that initiated the query); (ii) then, when both parameters (the connectivity and the number of synapses) have reached a certain threshold, the searches can touch more synapses, and the whole network becomes progressively visible, multiple parallel searches become more and more frequent and distant nodes (and keys) are reached faster. As we can see, increasing the number of synapses decreases the latency by only a small constant factor. In other words, synapse topologies do not need a lot of synapses to be efficient. This result fits with random graphs behavior: when the number of neighbors in the graph reaches a (small) threshold, the probability for the graph to be connected tends towards 1. Obviously, multiple searches in parallel lead to an increased number of messages. As illustrated in (top right), this number increases proportionally with the connectivity and the number of synapses.

Time-To-Live. As we pointed out, the number of messages can become high when the number of synapses increases. To limit this impact, we introduced a Time-to-Live (TTL) to reduce the overhead while keeping an acceptable level of exhaustiveness. We launched a second set of experiments in order to study the impact of the TTL on the search queries. This TTL is simply decreased every time the query traverses a node.

The purpose here is to preserve significant exhaustiveness, while reducing the amount of communications undergone by the inter-overlay. We made the number of overlays vary, to experiment with the impact of the *granularity* of the network. In other words, a Synapse network made of a few large structured intra-overlays could be called *strongly structured*, while another network with many smaller structured intra-overlays could be called *weakly structured*. The number of nodes was still set to 10000, and every node is a synapse belonging to 2 overlays chosen uniformly at random.

Figure 2 (mid left) confirms that a low synapse degree (2) is enough to achieve quasi-exhaustiveness. Another interesting result is that the TTL can be bounded without any impact on the exhaustiveness (10 or 12 is enough even when the number of overlays interconnected is 500), while, as highlighted (mid right), drastically reducing the amount of communications experienced, with the number of messages being almost divided by 2. To sum up, Synapse architectures can use TTL, leading to a significant exhaustiveness, while drastically reducing the expected overhead. Finally, the *granularity* (defined above) does not significantly influence exhaustiveness and communications when the number and connectivity of the synapses are fixed.

Connectivity and Peers' churn. Figure 2 (bottom left) shows the evolution of the exhaustiveness while increasing the average number of overlays a synapse belongs to. We repeated the experiment for different ratios of synapses (in percentage of the total number of nodes). The exhaustiveness is improved by increasing both factors. We obtain more than 80% of satisfaction with only 5% of nodes belonging to 10 floors, and other

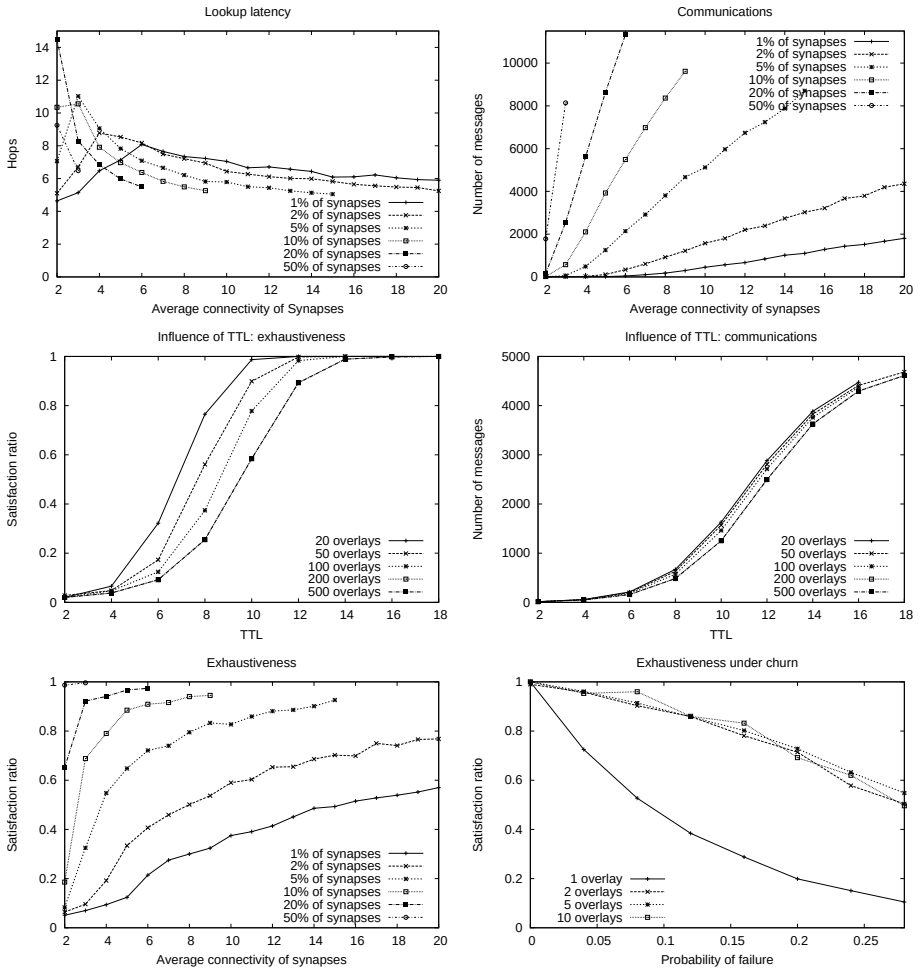

Fig. 2. Latency, communications, exhaustiveness and communications in Synapse

nodes belonging to only one intra-overlay. When each node belongs to 2 overlays, the exhaustiveness is, also, almost guaranteed.

Since networks are intended to be deployed in a dynamic settings (nodes joining and leaving the network without giving notice), we conducted a final set of simulations to see the tolerance of Synapse compared to a single Chord overlay network. In other words, the question is: *does an interconnection of small Chords better tolerate transient failures than one large unique Chord?* In this experiment, the number of nodes remained the same, whatever the topology tested. At each step, a constant fraction of nodes was declared temporarily unreachable (the failed nodes would reappear in the next step, thus simulating the churn). As a consequence, the routing of some messages failed. As highlighted (bottom right), improvement on the number of satisfied requests can be obtained through a Synapse network: when the probability of failure/disconnection of a node increases, the global availability of the network is far less reduced with Synapse

than with Chord. This shows that such synapse architectures are more robust and, thus, are good candidates for information retrieval on dynamic platforms. In other words, it demonstrates that this promising paradigm (interconnection of independent overlay networks) provides more fault-tolerance than a single global overlay network.

4 The Experimentations

JSynapse. In order to test our protocols on real platforms, we have initially developed `JSynapse`, a Java software prototype, which uses the Java RMI standard for communication between nodes, and whose purpose is to capture the very essence of our Synapse protocol. It is a flexible and ready-to-be-plugged library, which can interconnect any type of overlay networks. In particular, `JSynapse` fully implements a Chord-based inter-overlay network. It was designed to be a lightweight, easy-to-extend piece of software. We also provided some practical classes, helping with automating the generation of the inter-overlay network and the testing of specific scenarios. 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. Our goal here was to give a proof of concept and show the viability of the Synapse approach, while confirming results obtained by simulation about the overlay interconnection paradigm.

We used one cluster located at Sophia Antipolis, France. The `Helios` cluster consists of 56 quad-core AMD Opteron 275 processors linked by a gigabit Ethernet connection. 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). More precisely, for each number of networks (3 and 6), the number of nodes increases over time steps (10 to 50, and 100 to 150, respectively). One step was as follows: First, a set of queries is sent to the network (and results are collected). Second, some synapses are added for the next step.

We focus on the metrics affecting the user. Once his request was sent, a user waits for 1 second before closing the channels opened to receive responses. If no response was received after 1 second, the query is considered as not satisfied.

Figure 3 (left) shows the satisfaction ratio when increasing the number of synapses, for both the white and the black box version. As expected, the general behavior is comparable to the simulation results, and a quasi-exhaustiveness is achieved, with only a connectivity of 2 for synapses. Figure 3 (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 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.

Open-Synapse. We have also developed `open-synapse`, based on the stable and widely used `open-chord` implementation, which provides a complete and efficient Chord implementation. Open-Synapse extends the `open-chord` core, thus taking advantage of its robustness and reliability. A preliminary set of tests on `open-synapse` involved 50 nodes and different randomly generated scenarios.

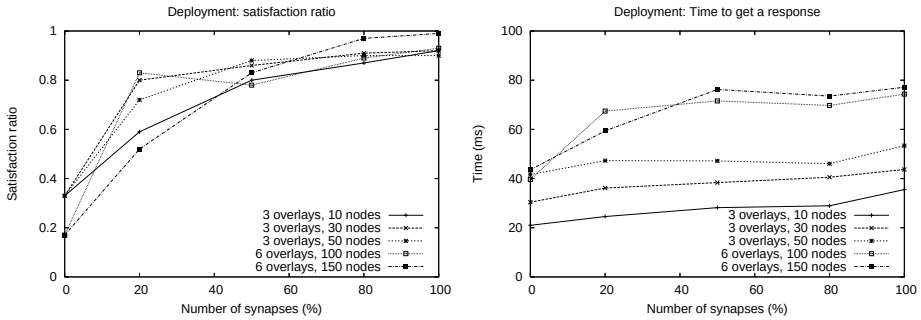

Fig. 3. Deploying Synapse : Exhaustiveness and Latency

5 Related Work

Cooperation through hierarchy. Several propositions have been made over the years to build topologies based on the coexistence of smaller local overlay networks. The first approach was based on hierarchical systems [6, 18], with some elected super-peers. In a more general view, merging several co-existing structured overlay networks has been shown to be a very costly operation [5, 16].

In the context of mobile ad hoc networks, Ariwheels [1, 12] was designed to provide a large variety of services through a multi-layer overlay network. Ariwheels provides an efficient mapping between physical devices in the wireless underlay network and virtual entities in the overlay network.

Cooperation through gateways. Authors in [4] present two models for two overlays to be (de)composed, known as *absorption* (a sort of merging) and *gatewaying*. Their protocol enables one CAN-network to be completely absorbed into another. They do not specifically take advantage of a simple assumption that nodes can be part of multiple overlays at the same time, thus playing the role of natural bridges.

More recently, authors in [3] propose a novel information retrieval protocol, based on gateways, called *DHT-gatewaying*, which is scalable and efficient across structured overlay networks[6]. They argue that there is not one preferred structured overlay network implementation, and that peers are members of co-existing DHTs. Although the focus is on wireless ad hoc network, they claim that their protocol can be used in wired networks too. Unfortunately, it is unclear how they evaluate their protocol.

Cooperation through co-located nodes. Authors in [11] present Synergy, an architecture which improves routing performance by providing cooperative forwarding of flows between networks. They try to create and maintain long-lived flows (*i.e.* long-lived paths) that can be used to cross overlay boundaries. However they do not go into details as much as we do in this paper regarding the algorithms that enable such overlay inter-connection.

Authors in [10] present algorithms which enable a symbiosis between different overlays networks with a specific application in mind: file sharing. They propose mechanisms for hybrid P2P networks cooperation and investigate the influence of system conditions, such as the numbers of peers and the number of meta-information

[6] Ex. Two 160-bit Chord, or two 160/256-bit Chord, or one 160-bit Chord and one 256-CAN.

a peer has to keep. Again, a more comprehensive understanding of this approach is missing. Authors only focus on file sharing whereas our protocol can be applied to any application. Even if the algorithms and the various observations they present are relevant, they fail to provide both any real experiments, and an in-depth analysis of their algorithms.

Authors in [7] consider multiple spaces with some degree of intersection between spaces, *i.e.* with co-located nodes. They focus on different potential strategies to find a path to another overlay from a given overlay, *i.e.* how requests can be efficiently routed from one given overlay to another one. They showed that with some dynamic finger caching, and with multiple gateways, they obtain pretty good performances. Their protocol focuses on the interconnection of DHTs, while we extend it to any kind of overlay.

In our previous preliminary work [13], we introduced BabelChord, a protocol for inter-connecting Chord overlay networks using co-located nodes that are part of multiple Chord "floors". These nodes connect, in an unstructured fashion, several Chord overlays together. The simulations showed that we could achieve pretty high exhaustiveness with a small amount of those co-located nodes. Our current paper, in turn, focuses on the co-located nodes heuristic in far more details than the aforementioned work, by providing not only a more generic protocol which enables inter-overlay routing that can in principle be applied to connect arbitrary heterogeneous overlays, but also more simulations to show the behaviors of such networks as well as a real implementations and live experiments.

Self-Aware Networks (SAN) and Cognitive Packets Networks (CPN). In SAN [9], nodes can discover new paths dynamically when the need to communicate arises; nodes can be able to discover new routes and optimize their QoS. CPN [8] are SAN where *smart packets* are used to discover new paths. A Synapse network is also a SAN but not a CPN. Every overlay keeps its proprietary routing and uses synapses nodes to explore other overlays that couldn't have been reached.

6 Conclusion

We have introduced Synapse, a scalable protocol for information retrieval in heterogeneous inter-connected overlay networks relying on co-located nodes. Synapse is a generic and flexible *meta*-protocol which provides simple mechanisms and algorithms for easy overlay network interconnection. We have given the set of algorithms behind our protocols and provided a set of simulations allowing to capture the behavior of such networks and show their relevance in the context of information retrieval, using key metrics of distributed information retrieval. We have also developed `JSynapse`, a lightweight implementation of Synapse, and experimented with it using the Grid'5000 platform, thus confirming the obtained simulation results and giving a proof of concept.

As future work, we first intend to focus on social mechanisms involved in synapses, which can greatly influence the shape of the network. On the practical side, we plan to extend `JSynapse` and plug in some other overlay network implementations. More intensive tests and deployments of `open-synapse`, our early prototype based on `open-chord` will also be considered. Adding CPN capabilities to Synapse networks would also be worth studying.

A "White" and "Black" box Synapse protocols

Figure 4 presents the pseudo-code of the two implementations of Synapse protocol using the message passing paradigm. For obvious lack of space, the code cannot be commented in great detail; the interested reader can refers to the web appendix.

The White Box protocol (first part) implements the same strategy to inter-route both `PUT` and `GET` requests and describes the social primitives `JOIN` and `INVITE` which are used to handle the invitation and join of new networks by a Synapse node.

The Black Box protocol (second part) on the other hand, is a set of messages to an independent Synapse Controller standing between the application layer and the actual overlay networks. The Controller can perform the following operations: (i) it can be notified by the application to initiate a new synapse GET/PUT, via the messages `SYN_GET`, `SYN_PUT`; in this cases it will take care of initializing the shared inter-routing data in the Control Network and fire a conventional GET/PUT message on one or more overlays it is connected to; (ii) it can receive a notification from one of the connected overlays about a GET/PUT request going through that could therefore be "synapsed" in the other networks. This is done via the call back methods `GET` and `PUT`. The Control Network operations are here represented for clarity as simple associative arrays. The notation `KeyTable[network.ID|hashKey] = key` is equivalent to send a `KEYTABLE_PUT` message with arguments `network.ID|hashKey` and `key` to the `SynapseControlNetwork`, while `key = KeyTable[network.ID|hashKey]` corresponds to send `KEYTABLE_GET` with argument `network.ID|hashKey` to `SynapseControlNetwork` and then to collect the result in the variable `key`.

References

1. D. Borsetti, C. Casetti, C. Chiasserini, and L. Liquori. *Content Discovery in Heterogeneous Mobile Networks*, in *Heterogeneous Wireless Access Networks: Architectures and Protocols*, pages 419–441, Springer-Verlag, 2009.
2. M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20, 2002.
3. L. Cheng. Bridging Distributed Hash Tables in Wireless Ad-Hoc Networks. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 5159–5163, 2007.
4. L. Cheng, R. Ocampo, K. Jean, A. Galis, C. Simon, R. Szabo, P. Kersch, and R. Giaffreda. Towards Distributed Hash Tables (De) Composition in Ambient Networks. In *Proc. of DSOM*, *LNCS* 4269, pages 258–268, Springer-Verlag,2006.
5. A. Datta and K. Aberer. The challenges of merging two similar structured overlays: A tale of two networks. In *Proc. of IWSOS*, 2006.
6. L. G. Erice, E. W. Biersack, K. W. Ross, P. A. Felber, and G. U. Keller. Hierarchical P2P Systems. In *Proc. of Euro-Par*, *LNCS* 2790, pages 1230-1239, Springer-Verlag, 2003.
7. P. Furtado. Multiple Dynamic Overlay Communities and Inter-space Routing. In *Proc. of DBISP2P*, *LNCS* 4125, pages 38–49, Springer-Verlag, 2007.
8. E. Gelenbe. Cognitive Packets Networks. U.S. Patent 6,804,201, October 2004.
9. E. Gelenbe. Steps Toward Self-Aware Networks. *Commun. ACM*, 52(7):66–75, 2009.
10. K. Junjiro, W. Naoki, and M. Masayuki. Design and Evaluation of a Cooperative Mechanism for Pure P2P File-Sharing Networks. *IEICE Trans Commun (Inst Electron Inf Commun Eng)*, E89-B(9):2319–2326, 2006.
11. M. Kwon and S. Fahmy. Synergy: an Overlay Internetworking Architecture. In *Proc. of International Conference on Computer Communications and Networks*, pages 401–406, 2005.
12. L. Liquori, D. Borsetti, C. Casetti, and C. Chiasserini. An Overlay Architecture for Vehicular Networks. In *Proc. of IFIP Networking*, *LNCS* 4982, pages 60–71. Springer-Verlag, 2008.
13. L. Liquori, C. Tedeschi, and F. Bongiovanni. Babelchord: a social tower of dht-based overlay networks. In *Prof. of IEEE ISCC*, pages 307–312, 2009.
14. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Adressable Network. In *ACM SIGCOMM*, pages 161-172, 2001.
15. A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-To-Peer Systems. In *Proc. of Middleware*, *LNCS* 2218, pages 329-350, Springer-Verlag, 2001.
16. T. M. Shafaat, A. Ghodsi, and S. Haridi. Handling network partitions and mergers in structured overlay networks. In *Proc. of P2P*, pages 132–139. IEEE Computer Society, 2007.
17. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup service for Internet Applications. In *Proc. of ACM SIGCOMM*, pages 149–160, 2001.
18. Z. Xu, R. Min, and Y. Hu. HIERAS: A DHT Based Hierarchical P2P Routing Algorithm. In *Proc. of ICPP*, pages 187, IEEE Computer Society, 2003.

```
White Box protocol
1.01 on receipt of OPE(code,key,value) from ipsend do      receive an opcode, key value from ipsend
1.02  tag = this.new_tag(ipsend);                          create a new unique tag for this lookup
1.03  send FIND(code,ttl,mrr,tag,key,value,ipsend) to this.myip;      send FIND msg to itself
2.01  on receipt of FIND(code,ttl,mrr,tag,key,value,ipdest)from ipsend do      rcv FIND
2.02   if ttl = 0 or this.game_over?(tag)      lookup aborted because of zero ttl or game over strategy
2.03   else this.push_tag(tag);                push the tag of the query as ''already processed''
2.04     next_mrr = distrib_mrr(mrr,this.net_list);          split all the maximum replication rate
2.05     for all net ∈ this.net_list do                      for all net the synapse belongs do
2.06       if this.isresponsible?(net,key)      the current synapse is responsible for the key in the net
2.07         send FOUND(code,net,mrr,key,value) to ipdest;      send a FOUND msg to ipdest
2.08       else if this.good_deal?(net,ipsend)            the net/ipsend is a ''good'' net/synapse
2.09              send FIND(code,ttl-1,next_mrr.get(net),tag,key,value,ipdest)
                    to this.next_hop(key);                 send a FIND msg with ... to ...
3.01 on receipt of FOUND(code,net,mrr,key,value) from ipsend do rcv FOUND msg from ipsend
3.02   this.good_deal_update(net,ipsend);          update my good deal tables with net and ipsend
3.03   match code
3.04    code=GET                                                                    GET code
3.05     send READ_TABLE(net,key) to ipsend                send a READ_TABLE msg to ipsend
3.06    code=PUT                                                                    PUT code
3.07     if mrr < 0                                    stop replication since no inter PUT is allowed
3.08     else send WRITE_TABLE(net,key,value) to ipsend    send a WRITE_TABLE msg (omitted)
3.09 on receipt of INVITE(net) from ipsend do      receive an invitation to join the net from ipsend
3.10   if this.good_deal?(net,ipsend)      the net/ipsend is a ''good'' net/synapse (left to peer's strategy)
3.11     send JOIN(net) to ipsend;                      send a JOIN message to reach the net to ipsend
4.01 on receipt of JOIN(net) from ipsenddo      receive a JOIN message to reach the net from ipsend
4.02   if this.good_deal?(net,ipsend)  the net/ipsend is a ''good'' net/synapse (left to synapse's strategy)
4.03     this.insert_net(net,ipsend);                   the current synapse insert ipsend in the net
Black Box protocol
1.04 on receipt of SYN_GET(key,cacheTTL,[targetNetworks]) from ipsend do   GET request
1.05 CacheTable[key].TimeToLive = cacheTTL;                                 init the control data
1.06 CacheTable[key].targetedNetworks = [targetNetworks]; specify the net to target if any specific
1.07 for all network ∈ (this.net_list ∩ targetNetworks) do
1.08  KeyTable[network.ID|network.hash(key)] = key;       store the unhashed key in the KeyTable
1.09  result_array += network.get(network.hash(key));      retrieve the result from each network
1.10  result_array += CacheTable[key].cachedResults; retrieve the cache content results from network
1.11  send SYN_FOUND(key,result_array) to ipsend;
2.01 on receipt of SYN_PUT(key,value,mrr) from ipsend do      a PUT is instantiated by a synapse
2.02 if (mrr > this.networks.size)  if MRR is bigger than the number of connected overlays for the synapse
2.03   mrrInSight = this.network.size;        compute how many replications are done in this synapse
2.04   mrrOutOfSight = mrr-this.networks.size;  compute how many replications are left after this step
2.05   delete ReplicationTable[key];                        refresh the replicationTable entry
2.06   ReplicationTable[key].ReplicasLeft = mrrOutOfSight;      update with new MRR value
2.07 else mrrInSight = mrr;                                  else we replicate for only MRR times
2.08 for i = [1:mrrInSight] do
2.09   KeyTable[this.net_list[i].ID|this.net_list[i].hash(key)] = key; store unhashed key
2.10   this.net_list[i].put(this.net_list[i].hash(key),value);  continue forwarding the PUT
3.01 on receipt of GET(hashKey) from this.net_list[i] do   a synapse rcv a GET to be forwarded
3.02 key = KeyTable[network.ID|hashKey];                  get the unhashed key from the Key Table
3.03 if (CacheTable[key] exists)                  if there is an entry in the Cache Table for the key
3.04  for all net ∈ (CacheTable[key].targetedNetworks ∩ this.net_list) do
                                          compute which of the specified networks we are connected to
3.05   KeyTable[net.ID|net.hash(key)] = key;                          add entry in KeyTable
3.06   results += net.forward_get(replicaNetwork.hash(key));          append the result
3.07  CacheTable[key].cachedResults += results;       store the collected results in the Cache Table
3.08  return results[1];                                  return only the first result
3.09 else return net_list[i].get(hashKey);       if there is no entry just forward in the current network
4.01 on receipt of PUT(hashKey,value) from this.net_list[i] do
4.02 key = KeyTable[network.ID|hashKey];                  get the unhashed key from the Key Table
4.03 if (ReplicationTable[key] exists)          if there is an entry in the Replication Table for the key
4.04  for all net ∈ this.net_list do
4.05   if (ReplicationTable[key].ReplicasLeft > 0) and     if there are still replicas and the network
4.06      not (ReplicationTable[key].hasNetwork?(replicaNetwork.ID))   hasn't had a PUT yet
4.07    KeyTable[net.ID|net.hash(key)] = key;                       add entry in KeyTable
4.08    ReplicationTable[key].addNetwork(net.ID);                   add the target network
4.09   ReplicationTable[key].ReplicasLeft--;          decrement the number of replicas to be done
4.10   net.forward_put();                             forward the message in the foreign network
4.11 else net_list[i].put(hashKey,value);       if there is no entry just forward in the current network
```

Fig. 4. The Synapse white and black box protocol