# From key-based to content-based routing : system interconnection and video streaming applications

Vincenzo Ciancaglini

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

ÉCOLE DOCTORALE STIC
SCIENCES ET TECHNOLOGIES DE L'INFORMATION
ET DE LA COMMUNICATION

T H È S E

pour obtenir le titre de

**Docteur en Sciences**

de l'Université de Nice - Sophia Antipolis
**Mention : Informatique**

Présentée et soutenue par

Vincenzo Ciancaglini

# From key-based to content-based routing: system interconnection and video streaming applications

Thèse dirigée par Luigi Liquori

préparée à l'INRIA Sophia Antipolis, Projet LogNet

soutenue le 26 Juliet 2013

**Jury :**

## Acknowledgments

This thesis marks the conclusion of a great adventure that I started four years ago, when I decided to give up the certainties of a well paid job as a software developer to take the path, completely unknown to me at the time, of a researcher.

I figured, therefore, that the best way to address my biggest thanks is chronologically.

Needless to say, I should begin thanking Luigi Liquori and Jean-Christophe Pazzaglia, my advisor and co-advisor for making it possible for me to start this new career of mine and finding the necessary fundings.

The first people I met when starting my studies where Laurent Vanni and Petar Maksimović. To Laurent goes all my gratitude for the great time spent together sharing an office and various alcoholic supplements to our coffee, and my best whishes for its new married life and its career. Mr. Maksimović is probably the main reason why most of my publications could boast a proper English syntax and avoid, as much as possible, that "maccarone" flavour so many publications from italian researchers feature (and most likely this acknowledgments too). Furthermore, he has not been just a collegue, he has become a friend, who has always been there for me to listen, encourage and help me throughout my work.

A special thanks goes to Alfredo Grieco, who is mostly reasponsible for the last part of this thesis, for giving me the opportunity of working with him and with his collaborators, Rossella Fortuna and Giuseppe Piro.

I should not even say it, but my family gets a special and honorable mention, having always supported me in ways that not everybody can praise. To my dad Gianfranco and my mom Marisa goes all my respect, my gratitude and my love.

While I approach the end of this adventure, looking forward at what lies ahead in my life and my career, one final thought of appreciation goes to the reviewers and members of the jury, who accepted to endure reading through the 160 pages of this work, a task that in my view requires abilities and an amount of patience no human creature could possibly have. Drawing the obvious conclusions from this fact, I, for one, welcome our new alien overlords and promise not to reveal their secret to our world leaders until the time comes.

In me they will find a loyal ally.

# Contents

# List of Figures

Chapter 1

# Introduction and scope of the thesis

**Contents**

## 1.1 Context of the thesis

Computers on the Internet, in order to communicate with one another, commonly rely on a host-based routing mechanism provided by the Internet Protocol (IP). Each machine is given a numerical address that uniquely identifies the network it belongs to and the node itself within the network. As such, the machine is able to send data packets to other machines in other networks by simply knowing their IP address. Routers on the edge of each of the networks ensure that packets directed to addresses outside the original network successfully reach their destination.

While the IP protocol is being satisfactorily used in most of the applications currently deployed on the Internet, the last few years have witnessed the development of a new class of distributed applications, where nodes in a network are arranged according to a logical topology, and do not anymore route the messages solely according to the address of the destination node to be reached, but, instead, the packets follow a routing path that also (or only) depends on the content they are carrying.

The fact that there exists a logical topology on top of the traditional IP layer, thus creating logical links with neighbor nodes not necessarily in the same IP network, led to dubbing such systems *Structured Overlay Networks*, in contrast with *Unstructured Overlay Networks*, in which nodes still create logical links with their neighbors, but not necessarily resulting in a specific overall organization.

For the purpose of this thesis we will not be covering unstructured overlays, which have been extensively studied in the literature, but rather focus our attention on structured overlays and the key-based routing mechanisms involved.

### 1.1.1 Structured overlays and key-based routing

Structured overlay networks fall in the category of peer-to-peer applications. However, unlike applications as BitTorrent [bit ], Gnunet [gnu ], or P2P-TV systems such as [nap ], where the collaborative aspect typical of the peer-to-peer paradigm is found in the joint distribution of content across the network, nodes of a structured overlay collaborate by being evenly responsible for a subset of resources to be managed.

Following and simplifying [Aberer 2005], the properties of structured overlays can be summarized as follows:

**Identifier space.** Nodes and resources are mapped onto a common identifier space $\mathcal{I}$, for example, the co-domain of a *consistent hash function* $H()$;

**Node identifiers.** Every node $N$ is assigned a logical identifier $ID_N \in \mathcal{I}$, using a mapping function $F_P()$. The mapping is often performed by hashing, with $H$, some unique property of the node, for example the IP and the port of the overlay network instance running on the node;

**Resource mapping.** Resources are also mapped into $\mathcal{I}$, using another mapping function $F_r$. This can be obtained by identifying them through a key $k$, which will be hashed so that $h_k = H(k) \in \mathcal{I}$;

**Space partitioning.** One or more nodes are held responsible for an interval of the addressing space $\mathcal{I}_N \subset \mathcal{I}$, often determined by a proximity function $d()$;

**Neighborhood graph.** Every node maintains a "view" of the whole network, in terms of pointers to other nodes in the system (i.e.its *network neighborhood*). This is one of the key factors that determines the *topology* of the network, also known as its *structuring strategy*.

While there are cases of fully connected overlays, where every node maintains pointers to every other node in the system, forming a fully connected topology, it is more often the case where nodes only discover and maintain a partial view of the network in the form of routing tables, the structure of which is another aspect characteristic of each different overlay network;

**Message routing.** Routing of messages in the network is driven by the resource identifier targeted by the message. For example, assuming that the goal of a message $M_k$ containing key $k$ is to reach one of the nodes $N_k$ responsible for the identifier space interval $\mathcal{I}_k$ where $H(k)$ falls in, a node with just a partial graph of the network may try to “get close” to $N_k$ by contacting the node in its routing table with the identifier closer to $N_k$, and have him repeat the operation until $N_k$ is reached. Hence the expression **key-based routing**;

**Overlay maintenance.** One final characteristic aspect of different overlays comes from their *Maintenance strategy*, i.e.the manner in which each node discovers and maintains the entries in its routing table.

In Figure 1.1, we give a better depiction of the ideas described above.

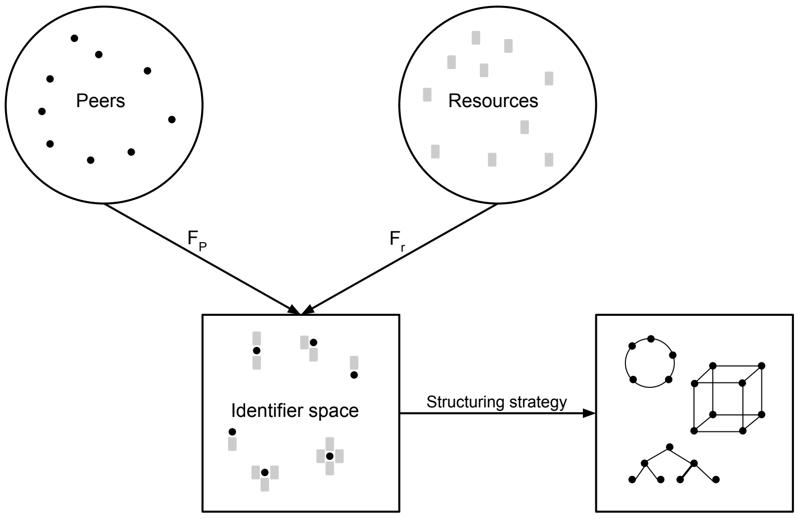


Figure 1.1: Structured overlay networks concepts, redrawn form [Aberer 2005]

It is worth mentioning that, when relying on a consistent hash function, as in the above example, to define and map data to an identifier space, we can also refer to overlay networks as *Distributed Hash Tables (DHT)*, which constitutes an efficient and scalable mechanism for resource lookup.

To further clarify the aforementioned points, here is a brief description of three of the most renowned overlay networks in literature, to better show the different approaches followed by researchers in designing structured overlays.

**Chord** [Stoica 2001], one of the first developed structured overlays, maps nodes and resource keys onto a ring spanning the co-domain of a consistent hash function. Each node $N$ is responsible for all of the keys $k$ falling into the interval $\mathcal{I}_N = ID_{N-1} < H(k) \leq ID_N$. In the most basic version, nodes maintain only a pointer to their successor, i.e. to that node with the lowest ID higher than $ID_N$, and routing works by forwarding a message for key $k$ form one successor to the other until the responsible node is found. However, to achieve logarithmic complexity in message routing, nodes also maintain a *Finger Table* of pointers to nodes responsible for the intervals identified by $ID_N + 2^i$, with $i$ being the index in the finger table, and during the routing, the message is forwarded to the node $NF$ in the finger table with the highest $ID_{NF} < H(k)$.

**CAN** [Ratnasamy 2001] is another algorithm developed with the same goals as Chord, but following a different implementation. Rather than being linear, the identifier space $\mathcal{I}$ is an $n$-dimensional toroid where each dimension is mapped using a separate hash function. Therefore, the nodes and the resources are mapped onto an $n$-dimensional Euclidean space, with nodes being responsible for a given partition of the space and resources being assigned to the node responsible for the partition they fall into. The routing table for a node $N$ consists of the list of nodes responsible for the coordinate zones adjacent to $\mathcal{I}_N$. Routing a message for key $k$ can follow different strategies, the simplest of which consists of sending the message to the neighbor node whose position in the space would make the message get the closest to the coordinates of $k$.

Using a different approach, **Kademlia** [Maymounkov 2002a] relies upon a XOR distance $d_{XOR}()$ as a proximity distance in its identifier space. Each node $N$ maintains a so-called *k-bucket list*, where a bucket $n$ contains a list of $k$ nodes $Nk$ whose identifier $ID_{Nk}$ has $n$ different bit from $ID_N$, i.e. $d_{XOR}(ID_N, ID_{Nk}) = n$. A node $N$ is responsible for a key $k$ if no other node in the system has a higher number of prefix bits in common with $H(k)$. Routing a message for a key $k$ by a node $N$ occurs in the following way: $N$ computes the number of matching prefix bit $n = d_{XOR}(ID_N, H(k))$, then selects, from the k-bucket $n$, $\alpha$ nodes to contact, in order to get nodes with an identifier closer than his to $k$, until no more closer nodes are found. Since $D_{XOR}()$ is symmetric, nodes do not necessarily need to issue maintenance messages to discover new entries for their routing tables, but can maintain them in an "opportunistic" way, by simply analyzing the undergoing requests for keys and storing the identifiers of the nodes issuing requests.

### 1.1.2 Applications of overlay networks

The scalability and lack of central control offered by structured overlays and distributed hash tables has been exploited in several application domains:

**Distributed databases.** For example, modern distributed databases such as *Cassandra* [Lakshman 2010] or *Dynamo* [DeCandia 2007] exploit a DHT-like key-value store to achieve horizontal scalability, being able to seamlessly add new machines to an existing server cluster and redistribute the charge without the need for complex master-slave setups.

**Anonymous networks.** Due to their lack of a centralized point of control, structured overlays are also at the foundation of many systems which guarantee anonymous communications over the Internet (*Darknets*), such as the *Onion Router* project (TOR) [tor ], as well as the *I2P Invisible Internet Project* [i2p ]. All of these rely on a DHT (CFS [Dabek 2001] in the case of TOR, Kademlia [Maymounkov 2002a] in the case of I2P) to implement mechanisms for discovery of resources and hidden services without the possibility for a third party to trace the underlying communications. Several other academic works, such as *Safebook* [Cutillo 2009], follow the same direction to implement secure and anonymous social networks.

**Resource discovery.** The Kademlia protocol is also renowned for being used as a support to initiate transfers in the BitTorrent network [Jimenez a], where a DHT is used to store and lookup the peers currently downloading the same content, ideally without relying on a centralized tracker to retrieve the information. With regard to this, it is interesting to note how, in fact, the very same BitTorrent network can rely on two similar but concurrent implementation of the Kademlia protocol, depending on the BitTorrent client in use. Other than that, several solutions to support resource allocation and discovery in Grid computing that rely on structured overlay networks have also been devised [Trunfio 2007].

**Search engines, Semantic Overlay Networks.** It is worth mentioning that overlay networks are not necessarily bound to a simple key-value store: recent research has brought to the development of entirely distributed search engines, such as Minerva [min ] and Yacy [yac ], that index more complex data on top of a distributed key-value structure. Furthermore, *Semantic Overlay Networks* [Crespo 2005] such as GridVine [Cudré-Mauroux 2007] allow for the storage of complex semantic data across a network of peers and the execution of complex queries in a distributed fashion.

**Other applications.** Other applications that fall outside the above category can include *Geographical Overlay Networks* [Ratnasamy 2003], where data from a sensors network is stored in a DHT according to its geographical proximity to a node, or *Distributed Virtual Worlds*, such as [Varvello ], where a Kademlia network has been exploited to store and retrieve information regarding a walkable virtual world such as, for example, *Second Life* [sec ].

### 1.1.3 Towards Content-Centric Networks

Recent advances in networking have led to the development of novel architectures where the IP layer can be entirely dropped in favor of a purely content-based approach in routing messages between nodes, known as *Content-centric networks* or *Networking named content* [Jacobson 2009b] [Zhang 2010].

Content-centric networks (CCN) are members of the so-called family of *Future Internet* applications. In a CCN, all content is unambiguously identified by a hierarchical name, allowing users to retrieve information without being aware of the physical location of the servers (e.g. IP address). CCN communications are driven by the consumer of data and only two types of messages are exchanged (namely *Interest* and *Data*). A user may ask for a content by issuing an *Interest*, which is routed within the CCN towards the nodes in possession of the required information, thus triggering them to reply with *Data* packets. The routing operations are performed by the strategy layer only for *Interest* packets. *Data* messages, instead, just follow the reverse path to the requesting user, allowing every intermediate node to cache the forwarded content. Because of this, an *Interest* does not necessarily need to reach the originator of the requested resource, but can be served by any node along the routing path that previously cached said content. This offers a natural congestion-control mechanism, gracefully reducing the propagation of messages for popular content. Furthermore, each node routing an *Interest* stores the pending request in a specific table, called *Pending Interest Table (PIT)*. By doing so, it can also reduce the propagation of multiple requests for the same pending content, by simply storing the origin of every interest for the same resource, and then routing the data back to each routing path from which an interest originated.

## 1.2 Problem definition

### 1.2.1 Interconnection of overlay networks

In the *Handbook of Peer-to-peer networking* [Shen 2010b], the overlay interoperability problem is identified amongst the research challenges related to peer-to-peer networks:

> **Protocols and interoperability:** Peers need to talk to each other. In some scenarios, peers belonging to different P2P overlays may also need to talk to each other. This requires well-defined protocols/interface, and a careful study of interoperability among P2P nodes.
>
> **Heterogeneity:** In reality, many aspects can affect the performance of P2P overlays, such as network availability/bandwidth, latency, peers' computational power and storage space, etc. Therefore, supporting heterogeneity is an important issue from a practical point of view.

In particular, when looking at the current state of overlay networks and their applications, we can think of the following opportunities:
**Exploiting locality to reduce the network size.** When faced with real-world conditions, many overlay protocols show severe performance degradation in terms of query response time, due to the impact that a high churn rate and network artifacts (e.g. the presence of NAT or firewalls between nodes) have on the communication symmetry in the individual node. While it is true that there is not a generic solution for these issues, many of them, being caused for example by communication timeouts, are directly related to the size of the overlay, as shown in works like [Jimenez a]. As such, it would be desirable to organize an overlay into smaller clusters connected together, possibly centered around some local property of the managed data, i.e.

language, topic, genre..., or the network itself, i.e. group together peers in the same country or behind the same Autonomous System.

This way, it should be possible to ensure faster delivery of messages within the same cluster for all the content accessed more often, while still maintaining the access to the rest of the network. With regard to this issue, it is worth mentioning the research carried on in the domain of *hierarchical overlays*, with works such as [Erice 2003a], [Erice 2003b], [Ganesan ] or [Xu 2003b].

**Exploiting heterogeneity.** Many overlay networks have been designed to best suit a specific kind of data (key-value pairs, geographical coordinates, semantic data), or specific class of nodes (server nodes, desktop machines, mobile terminals...). But, as of today, there is no solution good for all purposes. Applications (distributed or not) usually rely on *several* data structures and can span different classes of machines. With this in mind, it would be desirable to be able to design applications capable of accessing different, specialized overlay networks, without necessarily incurring the cost of having to maintain a connection to each one of them.

**Enabling cooperation.** There is an actual interest in the network interoperability itself, as a way to allow different existing applications to talk to one another. Works such as [Dabek ] and [Aberer 2005] have tackled the issue by trying to define and better formalize the various overlay protocols using a common framework.

**Case study: BitTorrent trackerless lookup.** The BitTorrent network and its trackerless system present a perfect study case to better illustrate the aforementioned points. According to [Jimenez a], the Kademlia network used in BitTorrent suffered from severely high response times (in the order of seconds, if not minutes) due to the high number of nodes in the network, the high churn and the timeout caused by the aforementioned network artifacts. Furthermore, the authors notice how there seem to be two different, competing implementations of the KAD network, incompatible with one another but serving the exact same purpose. Looking at this situation, the following considerations come to mind:

- In the impossibility of merging the two competing DHTs, it would be in the interest of the user for these two networks to talk to each other, since they share the same type of contents;

- There is a strong bias of requests for localized content in the network: Chinese-speaking users will more likely download contents in Chinese, and the same goes for any other language;

- With this in mind, it would be possible to improve the performance of the network, to group peers into smaller groups, sharing directly information only for localized content, while still leaving the possibility for a node to go out and scout for contents in foreign languages.

It is worth pointing out that, while the situation for the Kademlia network has greatly improved since [Jimenez a], also thanks to works like [Jimenez b], the principle exposed above still applies to a generic overlay.

In the present thesis we try to address the aforementioned problems by proposing an approach based on distributed gateways, i.e. nodes that, by connecting to several overlays at the same time, can re-route requests from one overlay to another. In particular, we tackle the problem by answering the following questions:

- **Performances:** How well would such a system perform? How resilient to real-world condition would it be?
- **Architecture:** What are the architectural and design challenges that arise when trying to design an interconnected system? How do we deal with different message encodings, protocols and routing schemes? How do we handle complex queries?
- **Tuning:** What is the optimal percentage of gateway nodes? How can we evaluate a million-node system in an efficient way?
- **Collaboration:** How can we deal with backward compatibility? A potential outcome of such an approach is the integration of existing, widely deployed overlays. Is it possible to handle their integration transparently?
- **Applications:** What are the possible uses for an interconnected system? What could motivate its development?

We will try, through our contributions, to give some answers to some of these questions.

### 1.2.2 Content-based techniques in real-time video streaming

Peer-to-peer real-time video streaming applications (P2P-TV) have notoriously neglected the adoption of content-addressable routing for data delivery, usually considering it only to initiate a transmission. P2P-TV systems usually rely on topologies where peers either build a hierarchical tree-like structure to diffuse the video chunks, or form an unstructured mesh-based network by building and maintaining a neighborhood of nodes with whom they exchange chunks of video, in a BitTorrent-like fashion. While the former is often hard to maintain and not very robust to churn, the latter approach shows better fault tolerance, but limits the possibilities a node has to retrieve a content only to its first degree neighbors, leaving it stranded if the required chunk of content is not found in time amongst its neighbors. With regard to this last issue, we can point out two key features of distributed hash tables:

1. The structured nature of their routing mechanism ensures that for each resource there is at least a peer holding it, making them suitable to retrieve rare contents;
2. The use of consistent hashing creates a statistically well-distributed traffic in the overall network;

It is therefore worth asking the question if such properties could be put to use in P2P-TV systems, as a way to providing each node with a mechanism to explicitly request a missing video chunk close to its expiration, in case it cannot be found in its mesh neighborhood.

Furthermore, the topic of content-addressable requests in real-time streaming is prominent when we think about Future Internet applications such as Content-Centric Networks. In a context where the concepts of "node" or "neighborhood" do not hold anymore, how would a real-time video distribution system behave? How effective are the caching mechanism of CCNs when dealing with short-living contents such as video chunks? How hard is to build a CCN-TV system, and what does it entail?

## 1.3 Outline of the thesis

In Part 1 we analyze the problem of overlay interconnection form a system design and modeling point of view.

Chapter 2 presents a first attempt to design a meta-protocol for network cooperation based on distributed gateways taking form of co-located nodes. The protocol, named Synapse [Liquori ], is a generic and flexible meta-protocol which provides simple mechanisms to efficiently route a request from one overlay to another. The behavior of said overlays is captured through an extensive set of simulations. Furthermore, the Synapse protocol has been implemented in JSynapse, a proof-of-concept developed in Java, and tested using the Grid5000 computing platform [Cappello *et al.* 2005], confirming the obtained simulation results.

This first work provided numerous insights about the practical challenges linked with overlay cooperation, such as communication security, routing of complex queries and backward compatibility with existing overlays. Said aspects are further analyzed in Chapter 3, where the Synapse protocol is extended ( [Ciancaglini 2012b] and [Ciancaglini 2012c]). Chapter 3 offers several contributions:

- A more detailed overview of the interconnection problem, that includes considerations about the security of links when exchanging sensitive data and an analysis of the backward compatibility problem with suggestions on possible workarounds;
- An exhaustive description of the novel Synapse 2.0 architecture, which is now able to transparently support several routing schemes and a more secure data exchange;
- A description of the development needed to implement the protocol in the OverSim Overlay Simulator [Baumgart ], that had to be heavily modified to support the instantiation of overlay nodes of different types;
- simulation results of the new protocol, in line with the results previously presented;

- results of experiments run on top of the Grid'5000 platform [Cappello *et al.* 2005], using a newly developed version of the Java client for Synapse.

Using simulation techniques and experimentation, albeit providing useful data on protocol behavior in both a controlled and a real-world scenario, enforces several limitations due to the scalability of said approaches, while, intuitively, a system made of interconnected overlays might shine best when deployed on a large-scale network, in the order of millions of nodes. But, how to tune such a system? How to determine the optimal number of gateway nodes, and the costs of such a deployment? To answer the above questions, in Chapter 4 we observe the problem under a different light, by providing a mathematical model to allow for the evaluation of interconnected systems ( [Ciancaglini 2012a] and [Gaeta 2013]). This model constitutes a first attempt to offer an estimate of the cost and performance, in terms of number of messages required and hit probability for a resource of arbitrary popularity, of a system made by a set of unstructured overlays connected by a given percentage of gateway nodes with a known connection degree. Model validation is provided thanks to a software developed from scratch in Erlang [erl ], showing the validity of our assumptions, and several examples are provided to show how it could be used to help the design of large-scale systems.

Part 2 offers a lighter interlude, where we present two proofs-of-concept of applications running on top of an interconnected system using the Synapse protocol. The first application, presented in Chapter 5 is called CarPal [Ciancaglini 2010], and it provides a way for nearby communities (i.e. schools, workplaces) to deploy a distributed database in order to share car rides and enable car sharing between their users. Through the use of gateway nodes, geographically close communities have the ability to extend the reach of their queries, thus increasing the chance to find a ride or a passenger. Chapter 6, on the other hand, deals with the problem of sharing document metadata in the field of archival and preservation of cultural heritage: there we show how a distributed system based on our architecture could leverage the power of commonly used desktop machines to create a database of document metadata that could be easily shared, amongst different institutions, thanks to the presence of distributed gateways [Marinković 2011].

In Part 3, we move away from the study of the overlays from a system point of view to a more general investigation into the possibilities offered by content-based routing in real-time multimedia applications. In Chapter 7 we outline a novel approach to peer-to-peer real-time video streaming (P2P-TV), where a structured overlay is used to support the NAPA-Wine P2P-TV system [nap ]. Through simulations, we thoroughly analyze the performance gain and message overhead of such a system, trying to determine whether such a solution would be optimal for real-world streaming. Finally, in Chapter 8, we move to am entirely new domain, developing a real-time video streaming application on top of Content-Centric Networks (CCN) [Jacobson 2009b]. CCN is a novel protocol for data routing over an existing IP network, and can also be viewed as a substitute for the IP protocol itself. As CCN is a protocol centered around content delivery, we analyze the requirements for developing, on top of it, an application dealing with expiring content (i.e. chunks of a video streaming), and how this might affect the behavior of the protocol.

Finally, in Chapter 9, we presents our conclusions and several directions for future work.

# Part I

# Design and modeling of interconnected overlay networks

Chapter 2

# Opportunistic routing on structured overlays

## Contents

In this chapter, we present our first attempt at tackling the problem of overlay cooperation, by designing and evaluating a novel meta-protocol, hereinafter referred to as *Synapse*, that allows for the information retrieval over the inter-connection of heterogeneous overlay networks.

Scalability in Synapse is achieved via the presence of co-located nodes (hence referred to as *Synapses*), i.e. nodes that are part of multiple overlay networks at the same time. By acting as a form of distributed gateways connecting several overlays, they offer a lightweight and scalable mechanism, which is *de facto* transparent to the applications sitting on top of the interconnected system.

Inter-overlay message routing is achieved without an explicit direct mechanism, but rather via an “opportunitic” behavior: nodes in overlay $A$ route messages using $A$’s routing algorithm; when a co-located node forwards a message, it can re-route it also in the other overlays it’s connected to, hence providing a mechanism to extend the search space for each node, without a specific logic being known to the internal nodes.

To succesfully perform inter-overlay routing, additional information needs to be provided to the co-located nodes, such as a time-to-live parameter, or a session ID to control the spread of packets in the whole interconnected system.

Synapse can work either with “open” overlays, where such parameters can be embedded in the overlay protocol packets, or with “closed” overlays, where due to compatibility issues the protocol cannot be modified to incorporate the required additional information. Furthermore, built-in primitives to deal with social networking give an incentive for nodes cooperation.

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 inter-connection of many Chord overlay networks.

## 2.1 State of the art

Pointing out the limits of a unique global structured overlay (rigidity, maintenance cost, security, ...), several proposition has been made over the years to build alternate topologies based on the co-existance of smaller local overlay networks. A first approach has been based on hierarchical systems ( [Erice 2003a], [Erice 2003b], [Xu 2003b] and [Ganesan ]), leading to the requirement of costly merging mechanisms to ensure a high level of exhaustiveness. In a more general view, merging several co-existing structured overlay networks has been shown to be a very costly operation [Datta 2006,Shafaat 2007], leading to inefficient overlay networks [Cheng 2006].

We organize the related work into two parts: related mechanisms that aim to enable/ease overlay inter-connection and some clean-slate routing architectures, as discussed in [Feldmann 2007], built from the ground up with networks interconnection in mind. Both parts share the same final goal, that is, providing easier ways to inter-connect networks. We could also cite some works that have been studying hierarchical DHT systems [Erice 2003a] which also consider multiple spaces and some elected super peers promoted to a top-level overlay network. But the main issue is that they introduce a multi-level addressing and lookup space whereas we, in this work, try to avoid it in order to be more generic. Hence we can say that our work subsumes hierarchical DHTs.

**Related mechanisms:** We can identify two main mechanisms for enabling overlay networks inter-communication on top of the current Internet architecture: *co-located nodes* and *gateways*. The main difference between the two is that co-located nodes fully participate in the routing process of the various overlays they are registered to, whereas gateways are nodes have only a specific pointer to another node in another overlay networks and do not actively participate in the routing process of the different overlays they are registered to. Co-located nodes have thus higher state overhead than gateways, since they have to maintain more pointers and process more messages due to their active participation. Yet, nowadays it is more common to use multiple P2P applications in the same time, and overlooking the possibility to exploit this would be limiting.

Recently, authors in [Cheng 2007], stating that complete merging is inefficient, propose a novel search protocol, based on gateways called "*DHT-gatewaying*", which is scalable and efficient across homogeneous[1], heterogeneous[2] and assorted[3] co-existing DHTs. Their argument is that there isn't a preferred DHT implementation, and that peers are members of co-existing DHTs. Their assumptions are (*i*) only some peers support the implementations of different DHTs and (*ii*) some peers are directly connected to peers that are members of other DHTs, and are called *Virtual Gateways (VG))*. Their gatewaying protocol works in the following way: when a request is sent in one DHT, and no result was found, the requester can decide to widen his search by forwarding its original search request to nodes which belong to other DHTs (cross-DHT search). Those nodes will "map" the search to the format which is supported by their relative DHTs. Once the mapping is done, the search is carried out in each DHTs, and if a result is found, it will be returned to the original requester. Note that a Time-To-Live (TTL) value is added to the original search, in order to avoid cycles; this value is decremented each time a request crosses a new DHT domain. Because VGs can be overloaded, authors devised a mechanism in order to distribute the mapping by electing more VGs (according to a specific VG determination scheme), and they also introduced self-organizing "gateways pointers" whose roles are to keep track of VGs where-abouts. Conceptually, this work seems the closest to our proposition. Our purpose is to give study more accurately ...

Author in [Furtado 2007] presents mechanisms for managing the multiple identifier spaces as well as inter-space linking and routing alternatives. They consider multiple spaces with some degree of intersection between spaces, i.e. with co-located nodes. They compared various inter-space routing policies by analyzing which trade-offs, in terms of state overhead, would give the best results in terms of the number of messages generated and routed, the number of hops it takes to find a result and

[1]Homogeneous DHTs: same implementation and same keysize (ex. Two 160-bit Chord DHTs)

[2]Heterogeneous DHTs: same implementation and different keysize (ex. One 160-bit Chord and one 256-bit Chord DHTs)

[3]Assorted DHTs: different implementation and/or different keysize (ex. One 160-bit Chord and one 256-CAN DHTs)

the state overhead (i.e. the number of fingers a node has to keep). They do not present any algorithms but they do provide an comparative analytical study of the different policies. They showed that with some dynamic finger caching and with multiple gateways (in order to avoid bottlenecks and single points of failures) which are tactfully laid out, they attain pretty good performances.

In [Cheng 2006] authors presented two models for two overlays to be (de)composed, known as *absorption* (equivalent to merging) and *gatewaying*. Their protocol enables a CAN-DHT to be completely absorbed into another one (in the case of the absorption), and also provide a mechanism to create bridges between DHTs (in the case of the gatewaying). They do not specifically take advantage of a simple assumption that nodes can be part of multiple overlays in the same time thus playing the role of natural bridges. They did not evaluate their protocol and do not provide any algorithms of their protocol.

Authors in [Junjiro 2006] present a model which considers the symbiosis between different overlays networks with a specific goal in mind: file sharing. They propose a mechanism for hybrid P2P networks cooperation and investigates the influence of system conditions such as the numbers of peers and the number of meta-information a peer has to keep. Their work is bit more generic in the sense that they do not focus on structured overlay networks as we do, but still, they provide interesting observations on: *(i)* joining a candidate network (i.e. considering to enhance one's QoS by joining another network), *(ii)* selecting cooperative peers (that is which peer(s) among this newly joined network will cooperate with me), *(iii)* finding other P2P networks, *(iv)* the very decision of starting cooperation, by taking into account the size of the network (for instance a very large network will not really benefit from a cooperation with a small network), *(v)* relaying messages and files, *(vi)* caching mechanisms in cooperative peers and finally *(vii)* when it is appropriate to end a cooperation. Their simulations showed the effect the popularity of a cooperative peer on the search latency evaluation, that is the more a node has neighbors, the better, as well as the effect of their caching mechanism which reduces (when appropriately adjusted) the load on nodes (but interestingly does not contribute to faster search).

Authors in [Kwon 2005] presents Synergy, an overlay inter-networking architecture which improves the routing performance in terms of delay, throughput and loss packets by providing cooperative forwarding of flows. Authors acknowledge that co-located nodes can serve as good candidates for enabling inter-overlay routing and that they reduce traffic.

In this work, and in a previous preliminary work [Liquori 2009], it is also argued that co-located nodes are also good candidates for widening the search capability. However here we focus on the co-located nodes heuristic in more details than the aforementioned works by providing not only a simple algorithm which enables inter-overlay routing but also more intensive simulations to show the behaviours of such networks as well as a real implementation and experiments. We first want to grasp the complete potential that co-located nodes offer and we want to deepen the study of overlay networks with these types of nodes.

**Clean-slate routing architectures:** The following works, although not directly related to ours, propose alternatives to the current Internet architecture and also present interesting methods for inter-connecting domains.

Authors in [Caesar 2006] propose and analyze a routing scheme based on flat names. They want to get rid of location information that we can find the network layer and route directly on the identities themselves. Although they propose a compact routing scheme, some questions arise regarding the scalability of their solution.

In [Yang 2003] authors present the design of a new Internet routing architecture (NIRA) that aims at providing end users the ability to choose the sequence of Internet service providers a packet traverses at the domain level, i.e. they will be able to choose how inter-domain routing is done. Authors argue that overlay networks are not ubiquitous, that only nodes on the overlay network can control the packet's paths by tunnelling traffic through other nodes on the overlay. They also present scepticisms regarding the scalability of the overlay, stipulating that they are unlikely to scale up so to include every user on the Internet, and that an overlay path may traverse duplicate physical links.

In [Zhu 2003] authors propose a routing scheme which separates structural information and dynamic information. They provide a system in which only structural information is disseminated, and dynamic information is discovered by routers based on feedbacks and probes, which apparently helps improving the routing decisions. Authors believe that overlay network is not the final solution for reliable packet forwarding. Their reasoning is based on the fact that overlay network only increase the probability that the communication does not fail when there are only isolated routing failures in the network. No overlay network is going to function when the underlying routing infrastructure completely fails.

Regarding the clean-slate redesigns of the Internet, most of the cited authors ( [Yang 2003], [Zhu 2003]) seem to agree that the BGP routing protocol [Rekhter 1995], the main protocol for inter-domain routing, does not provide enough information regarding the packet routes and does not give the possibility to the users to be able to choose their own domain-level routes. BGP does not scale particularly well, converges rather slowly (and sometimes with certain routing policy combinations it diverges [Labovitz 2000]). They attempt, and so do we, to circumvent the current Internet limitations by proposing an alternative method for interconnecting networks.

Although their insights and proposals are more than relevant, we do believe they are far from being applicable in practise. The obvious reason is that the current established Internet cannot be changed in such radical ways and their solutions cannot be easily deployed. In this sense, overlay networks are a more flexible solution than complete re-designs, plus they can also serve as a framework for clean-slate re-designs to accelerate prototyping their new approaches.

In this work we focus our attention on inter-connecting overlay networks, because we believe that since their introduction they have matured and they can answer most of todays Internet's challenges. We provide what we consider as a simple and natural solution for bridging overlay networks together.

In this sense, and in response to overlay detractors, we argue that works like [Xu 2003a], [Garces-Erice 2003] and [Zhou 2003] show efficient method for constructing an overlay network while taking into account the underlying topology. Therefore we can say with confidence that we do have mechanisms in order to ensure that the paths the packets traverse are not using duplicate physical links.

## 2.2 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. 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 about 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. 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 according to the policy of each of its intra-overlays, but will have the extra-role of forwarding the requests to some intra-overlay it belongs to.

- *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 belongs 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 keys distribution, we could say that also 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 responsibles of a specific set of resources. This invariant allows for routing under structured, unstructured and hybrid networks: the rationale is simple: by construction, intra-routing is the one always responsible for its correctness, since Synapse just cares about overlay's inter-connection.

- *Network independency 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 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 help the choice of whether or not to join another intra-overlay, invite or accept a peer to one of the overlays, 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 to 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 to 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 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.

**White box synapse.** The white box hereby presented 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, 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 keysize) 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 under-

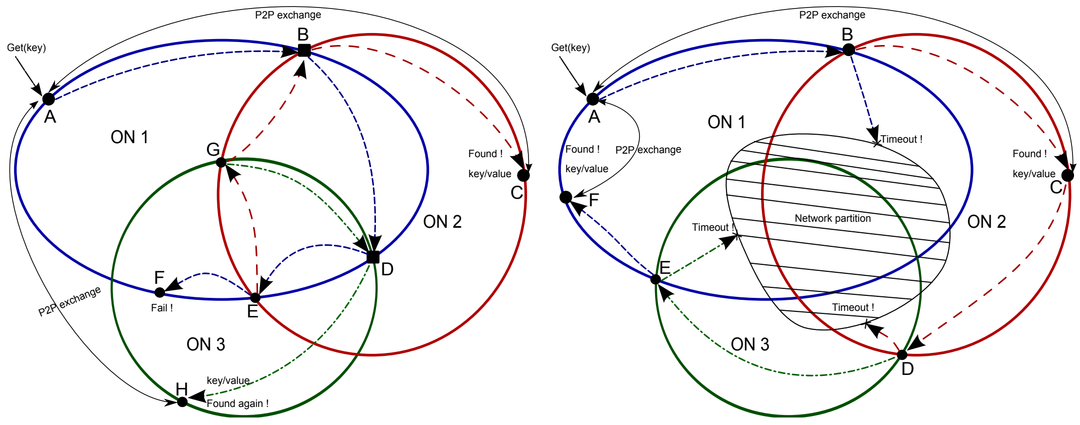


Figure 2.1: Routing across differents overlays and dealing with a network partition

going 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 is stored the number of times the key should be replicated across all of the the overlays; (*iii*) a Cache table, used to implement the replication of `GET` requests, and cache multiple responses and control the flooding of foreign networks.

**Example 1. Routing across differents intra-overlays.** Figure 2.1 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 the 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 the peer A and peer C responsible of 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 the 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 (again game over strategy). Peers playing game over strategy are depicted as squares.

**Example 2. Dealing with network partition.** Figure 2.1 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 alternative 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. The reader can refer to the web appendix for a detailed description of the protocol pseudocode, in both the white and the black box model.

## 2.3 “White box” Synapse protocol definition

```
1.01 on receipt of OPE(code,key,value) from ipsend do
1.02  tag = this.new_tag(ipsend);
1.03  send FIND(code,ttl,mrr,tag,key,value,ipsend) to this.myip;

2.01 on receipt of FIND(code,ttl,mrr,tag,key,value,ipdest)from ipsend do
2.02   if ttl = 0 or this.game_over?(tag)
2.03   else this.push_tag(tag);
2.04     next_mrrs = distrib_mrr(mrr,this.net_list);
2.05     for all net ∈ this.net_list do
2.06       if this.isresponsible?(net,key)
2.07         send FOUND(code,net,mrr,key,value) to ipdest;
2.08       else if this.good_deal?(net,ipsend)
2.09                send FIND(code,ttl-1,next_mrr.get(net),tag,key,value,ipdest)
                  to this.next_hop(key);

3.01 on receipt of FOUND(code,net,mrr,key,value) from ipsend do
3.02   this.good_deal_update(net,ipsend);
3.03   match code
3.04    code=GET
3.05     send READ_TABLE(net,key) to ipsend
3.06    code=PUT
3.07     if mrr < 0
3.08     else send WRITE_TABLE(net,key,value) to ipsend

4.01 on receipt of INVITE(net) from ipsend do
4.02   if this.good_deal?(net,ipsend)
4.03     send JOIN(net) to ipsend;

5.01 on receipt of JOIN(net) from ipsend do
5.02   if this.good_deal?(net,ipsend)
5.03     this.insert_net(net,ipsend);
```

Figure 2.2: The Synapse white box protocol

Figure 2.2 presents the pseudo-code of the protocol using message passing paradigm.

### 2.3.1 The `GET` operation

The `GET` operation consists in finding the value of an object we are seeking, provided its key. A node seeking an object sends an `OPE(GET,key,_)` message to an arbitrary node it knows. On receipt (see lines 1.01-1.03 ), the node generates a new tag `tag` for this request that will be associated with the query all along its path. The routing is then initiated with a given TTL by sending an auxiliary `FIND` message for this request to the node itself; this message seeks the node(s) responsible for the key sought in order to read the value (if it exists). Upon receipt of a `FIND` message, a node checks first if the TTL is valid and second if this query was already processed on the node: in both cases, the routing aborts, in order to avoid useless message overhead.

On receipt of a `FIND` message (see lines 2.01-2.09), the node checks the TTL and the tag of the request before starting processing the request, and, first, recording it as *already processed* ("game over" strategy). The retrieval process starts then locally, in two steps for each intra-overlay the node belongs to: (*i*) it checks if, according to the particular retrieval algorithm of the intra-overlay, it is itself assigned a range of keys containing `key` (line 2.06); if this is the case, for this overlay, the retrieval process ends and a `FOUND` message is sent back to the initiator of the request informing it that the potential value sought is stored on this node (line 2.07). (*ii*) if the node was not responsible for the key in this particular overlay, it forwards the request to the *next hop* inside this intra-overlay, according to the particular overlay's policy (line 2.09).

On receipt of such a `FOUND` message — recall that several responses can be obtained for a request — the initiator of the `GET` request sends a `READ_TABLE` message to the responsible of the key, basically to first to check if any value is assigned to this key and then to retrieve the value(s) and then get the value of the key sought (see lines 3.04-3.05).

### 2.3.2 The `PUT` operation

The `PUT` operation is a declaration of a resource. Depending on the purpose of the resource aggregation, the `PUT` policy may change:

- If the purpose of the aggregation is to let each overlay keep the control on their information (with exclusive rights for writing and updating the information) while letting nodes from other overlay read this information, the `PUT` operation will be performed independently within each overlay, each node declaring their resources to their intra-overlays. In this first case, the `PUT` operation will not be different as in a set of intra-overlays without inter-connection, and corresponds to set the Maximum-Replication-Rate (MRR) to 0.

- If the purpose of the aggregation is to build a globally distributed information system, each node may declare its resources to a set of intra-overlays it may not belong to. In this last case, the `PUT` operation involves mechanisms very similar to the `GET` operation and the Maximum-Replication-Rate (MRR) different than zero tells how many copies we want to distribute in the inter-overlay. Line 2.04 computes via the function `distrib_mrr` the required values of MRR for a `PUT` request, starting from both its current value and the number of intra-overlays the request will be forwarded to. Recall that MRR is ignored when the message is not a `PUT` operation. In fact, a node declaring a resource will also seek nodes in the Synapses responsible for their resources. Once such location is found (similarly than for the `GET` operation), the initiator of the request has just to send the value to be stored by the responsible nodes found. This is achieved by lines 3.06-3.08.

### 2.3.3 The `JOIN` and `INVITE` operations

The `JOIN` message is sent by a node entering the network, upon a reception of an `INVITE` message. Please refer to lines 4.01-4.03 for invitation, and to lines 5.01-5.03 for join. The intra-overlays in which a joining node will act can be chosen in different ways. A peer receiving an invitation to join a network through the `INVITE` message sent by another node can evaluate, via the `good_deal?` social-based primitive, the relevance of this invitation. If the invitation was positively evaluated, it can send a `JOIN` message to the peer that launched the invitation. Upon receipt of a `JOIN` message, a peer can decide, again via the `good_deal?` primitive, whether or not this join is interesting for it.

## 2.4 "Black box" Synapse Protocol definition

Figure 2.3 presents the pseudo-code of the protocol using the Black Box paradigm.

### 2.4.1 Accessing blackbox networks

The Synapse protocol hereby presented 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, so does the unhashed key in order to be rehashed when a synapse is met.

Interconnecting existing overlay made of "blind" peers, who are not aware of the additional parameters, seems one natural evolution for the synapse model 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 member 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.

### 2.4.2 Data structure

As a general operational model we can imagine a set of entities responsible for the interaction with the individual overlays at level N that communicate with a Synapse Controller at level N+1 through a set of primitives. The Synapse Controller access the routing information through the Control Network and is completely agnostic of the underlying protocols. 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:

#### 2.4.2.1 Key Table

When a node in a structured overlay issues a message related to a specific key $K$, it hashes $K$ using a hash function $H()$ specific to the overlay itself, and then issue a message containing only the hashed version $H(K)$. Since $H(K)$ is assumed to be non reversible, the Key Table is responsible for storing the unhashed version $K$ of the keys circulating in the underlying overlays. When a synapse node performs a `PUT` or a `GET` that it wishes to be replicated in other networks, it makes the unhashed key $K$ available to the other synapses through the Key Table. The key is stored using an index formed by a network identifier $NID$ as a prefix and the hashed key $H(K)$ as a suffix. This way, every other synapse in the originating network can easily retrieve $K$ using only the information they are aware of from the underlying messages (namely $H(K)$ and $NID$.)

In order to avoid that its size explodes a mechanism of local FIFO is envisioned for the Key Table. Each node of the Control Network should treat its part of data as a FIFO of fixed size, treating every new access to an item as an insertion thus preserving the items most wanted.

#### 2.4.2.2 Replication Table

The Replication Table is used to enable consistency during the replication of `PUT` messages across networks. When a synapse node performs a new `PUT` with replication, it inserts the unhashed key $K$ key in the Key Table and a new entry in the Replication Table in the form

`[H(K),mrr,ttl,[netid]].`

When another node receives the message to be forwarded, in order to perform a `PUT` in the other overlays it first checks if the `mrr` counter $> 0$. In case it performs a maximum of `mtt` replication of the `PUT` request, and decrements the `mrr`. To avoid sending the same request more than once in the same network, the Replication table stores a list of networks where the request has already been performed. A `ttl` parameter, set by the node issuing the `PUT` request, manages the expiration of the entry in the table and avoids the risk of having infinite loops due, for example, to an `mrr` set much higher than the number of overall networks and therefore never getting down to 0. In case of overlapping `PUT` requests of the same key by different synapses, a FIFO criterion is applied and the old entry in the table is completely overwritten by the new request parameters. However it should be mentioned that the replication of `PUT` requests across multiple networks is a critical point that need further investigation due to the many drawbacks, of top of whom is the problem of guaranteeing data consistency across networks in case of a new put of an existing key (data update).

#### 2.4.2.3 Cache Table

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(K),ttl,[netids],[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 `[netids]` networks he is connected to, or in all his networks if no `[netids]` are specified. All the responses are stored in the `[cache]` and only one is forwarded back, in order to maintain backward compatibility with possible blind nodes having performed the same request. As in the Replication Table, 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.

### 2.4.3 Algorithm

Hereby we present the algorithm adopted by the Synapse Controller to perform multiple `PUT` or `GET` in a set of network. The different approach to the problem compared to a White box model brings some limitations to certain functionality (e.g. request tagging is not possible) but allows on the other hand to implement additional options in the requests (e.g. selective broadcast during a `GET` request). The algorithm is described through the primitives exposed by a Synapse Controller to the upper and lower level. For simplicity all the operation performed on the Control Network's DHT are represented as local map operations, and are assumed to be synchronous. For example, `KeyTable[key]` correspond to a **send** `KeyTableGET(key)` **to** `ControlNetwork`. The implementation of the Control Network (choice of routing, topology ...) is not discussed here. To the upper level a Synapse Controller exposes the message `SYN_GET` and `SYN_PUT`, while to the lower level the Synapse Controller can exchange canonical `GET/PUT` messages with the entities responsible of the connection to the overlays.

- `SYN_GET` initiate a multiple `GET` operation in all networks. The parameters passed are the key to be searched, the Time To Live for the data in the Cache Table (this represents as well the duration of the flooding across the networks) and optionally a list of specific networks to target. Before sending the `GET` request to the networks the synapse is connected to, it initialize the Cache Table by adding a new entry with the specified `ttl` and the list of target networks if present. Then multiple requests are dispatched, taking care of storing for each network a copy of the unhashed key in the Key Table. When all the responses are received, the synapse collects also all the results stored in the cache, representing the responses from network out of direct sight.

- `SYN_PUT` initiate the data in the control network to perform multiple `PUT` requests. To begin with, the request is replicated to the first `mrr` networks to which the synapse is connected to. In case the `mrr` is higher that the number of connected networks (thus needing replication on out of sight networks), a new entry in the replication table is stored (or an old entry for the same key is replaced) with the remaining number of replications to do. Then, as per `SYN_GET`, the request is dispatched to the underlying networks, taking care of storing for each network a copy of the key in the Key Table.

- `GET` represents a `GET` request passing by to be replicated by the synapse. In order to replicate it, the Controller checks first if a copy of the unhashed key is stored in the control network (meaning that the request was initially performed by another synapse). If present, the Cache Table is checked to see if there is an entry corresponding to the requested key. If so, the controller dispatches the request either on the target networks it's connected to (if specified in the Cache Table) or to all its networks. To avoid breaking the compatibility with possible blind peers being able to handle only one response per request, only the first result is returned and the rest is stored in the Cache Table for later retrieval.

- `PUT` represents a passing `PUT` request to be replicated. As for the `GET`, the algorithm first retrieves the unhashed key for the network and, if present, the corresponding entry in the Replication Table. If there is such entry, the request is replicated in the networks not yet marked in the Network List corresponding to this entry, decrementing `ReplicasLeft` each time until 0 is reached. To avoid performing the request twice in the same network, the network ID is stored in the Network List.

## 2.5 The Simulations

The purpose of the simulations is to allow for better understanding of the behavior of structured overlay interconnection through the Synapse approach. We focus on the key metrics traditionally considered in 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). The goal is to highlight the behavior of these metrics while varying the topology (number of synapses and their connectivity, TTL, number of intra-overlays ...).

```
6.01 on receipt of SYN_GET(key,cacheTTL,[targetNetworks]) from ipsend do
6.02 CacheTable[key].TimeToLive = cacheTTL;
6.03 CacheTable[key].targetedNetworks = [targetNetworks];
6.04 if not (targetNetworks)
6.05  targetNetworks = this.networks;
6.06 for each network in (this.networks ∩ targetNetworks)
6.07  KeyTable[network.ID|network.hash(key)] = key;
6.08  result_array += network.get(network.hash(key));
6.09  result_array += CacheTable[key].cachedResults;
6.10  send SYN_FOUND(key,result_array) to ipsend;

7.01 on receipt of SYN_PUT(key,value,mrr) from ipsend do
7.02 if (mrr > this.networks.size)
7.03   mrrOutOfSight = mrr-this.networks.size;
7.04   mrrInSight = this.network.size;
7.05   delete ReplicationTable[key];
7.06   ReplicationTable[key].ReplicasLeft = mrrOutOfSight; )
7.07 else
7.08   mrrInSight = mrr;
7.09 for i = [1:mrrInSight]
7.10   KeyTable[this.networks[i].ID|this.networks[i].hash(key)] = key;
7.11   this.network[i].put(this.networks[i].hash(key),value);

8.01 on receipt of PUT(hashKey,value) from this.network[i] do
8.02 key = KeyTable[network.ID|hashKey];
8.03 if (ReplicationTable[key] exists)
8.04  for each replicaNetwork in this.connectedNetworks
8.05   if (ReplicationTable[key].ReplicasLeft > 0)
8.06    and not  (ReplicationTable[key].hasNetwork?(replicaNetwork.ID))
8.07    KeyTable[replicaNetwork.ID|replicaNetwork.hash(key)] = key;
8.08    ReplicationTable[key].addNetwork(replicaNetwork.ID);
8.09    ReplicationTable[key].ReplicasLeft--;
8.10    replicaNetwork.forward_put();
8.11 else
8.12  network[i].put(hashKey,value);

9.01 on receipt of GET(hashKey) from this.networks[i] do
9.02 key = KeyTable[network.ID|hashKey];
9.03 if (CacheTable[key] exists)
9.04  if (CacheTable[key].targetedNetworks is empty)
9.05   targetNetworks  = this.networks;
9.06   replicaNetworks = CacheTable[key].targetedNetworks ∩ this.connectedNetworks;
9.07  for each replicaNetwork in replicaNetworks do
9.08   KeyTable[replicaNetwork.ID|replicaNetwork.hash(key)] = key;
9.09   results += replicaNetwork.forward_get(replicaNetwork.hash(key));
9.10  CacheTable[key].cachedResults += results;
9.11  return results[1];
9.12 else
9.13  return network[i].get(hashKey);
```

Figure 2.3: The Synapse blackbox protocol

### 2.5.1 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 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 used 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 hop. Each result is the average of 50 simulation runs.

### 2.5.2 Impact of Synapse nodes and their connection degree

Our first set of simulations has 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). The number of nodes was fixed to 10000, uniformly distributed amongst 20 overlays (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 among 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 to obtain the first successful response. As illustrated in Figure 2.4, one first point to notice is that the number of hops remains logarithmic when changing a Chord network into a Synapse network (the number of nodes is 10000, the latency never exceeds 14). Other experiments conducted by increasing the number of nodes confirm this. More precisely, Figure 2.4 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 of only a small constant factor. In other words, synapse topologies does 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 Figure 2.5, this number increases proportionally with the connectivity and the number of synapses.

### 2.5.3 Effects of 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

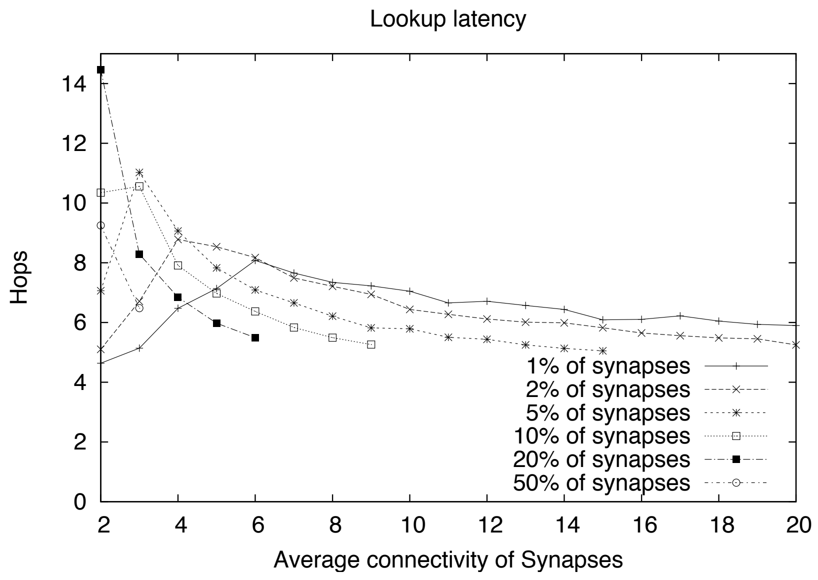


Figure 2.4: Latency in Synapse

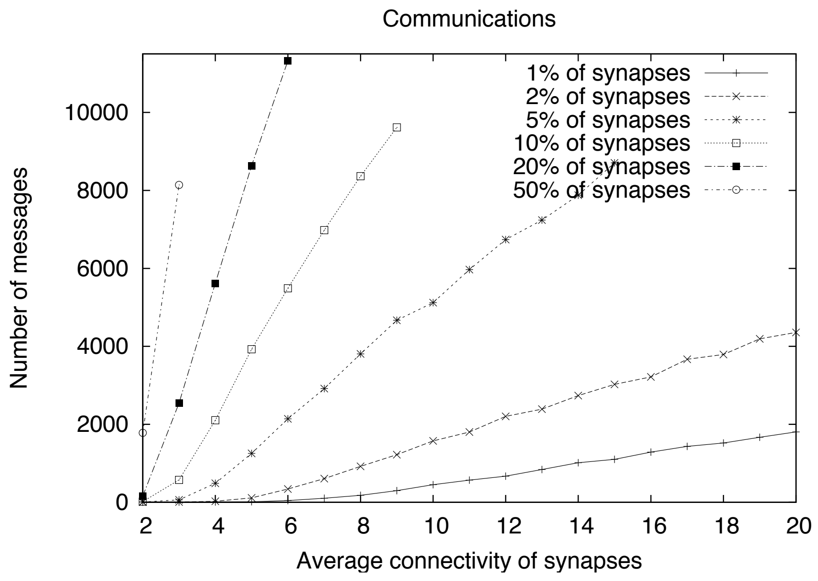


Figure 2.5: Communications overhead in Synapse

node.

The purpose is 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 the impact of the *granularity* of the network. In other words, a Synapse network made of 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

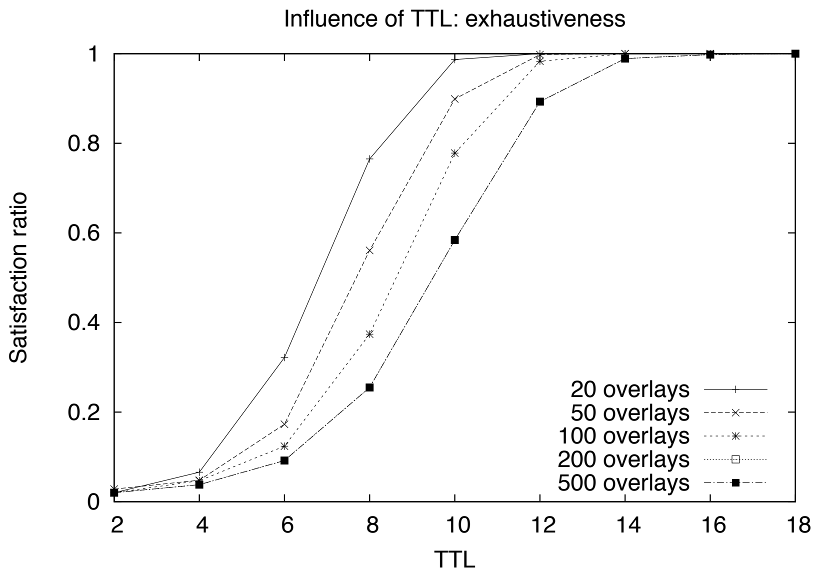


Figure 2.6: TTL vs. exhaustiveness

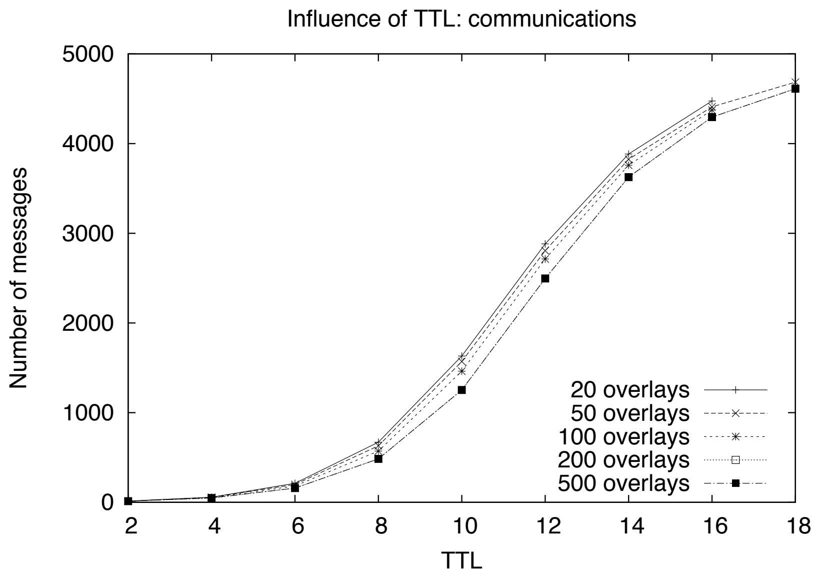


Figure 2.7: TTL vs. communications

to 10000, and every node is a synapse belonging to 2 overlays chosen uniformly at random.

Figure 2.6 confirms that a low synapse degree (2) is enough to achieve quasi-exhaustiveness. Another interesting result is that 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 by Figure 2.7, drastically reducing the amount of communications experienced, with the number of messages being almost

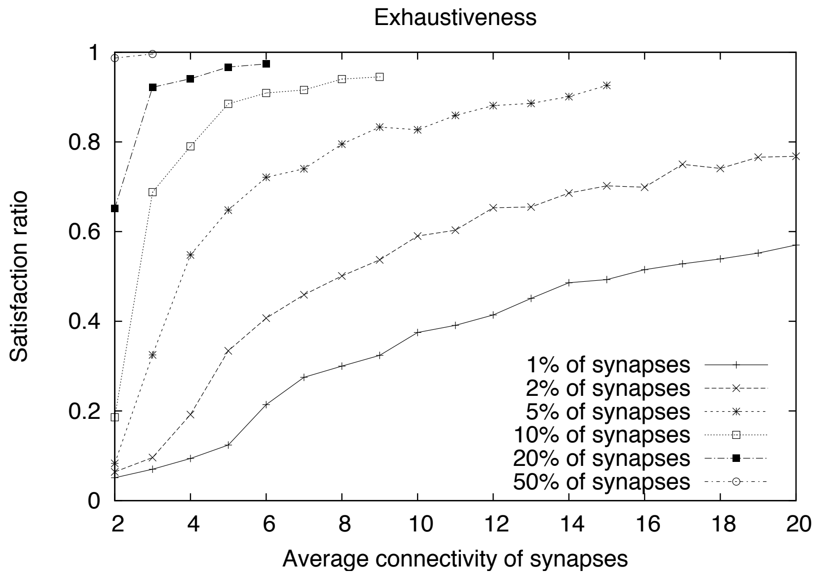


Figure 2.8: Exhaustiveness vs. synapse connectivity

divided by 2. To sum up, Synapse architectures can use TTL, leading to a significant exhaustiveness while drastically reducing the expected overhead. Finally, still see Figure 2.6, the *granularity* (defined above) does not significantly influence exhaustiveness and communications when the number and connectivity of the synapses are fixed.

### 2.5.4 Connectivity and Peers' churn

Figure 2.8 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 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, at each step, a subset of nodes is declared unreachable (simulating the churn), making message routing fail. As we can see on Figure 2.9, 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

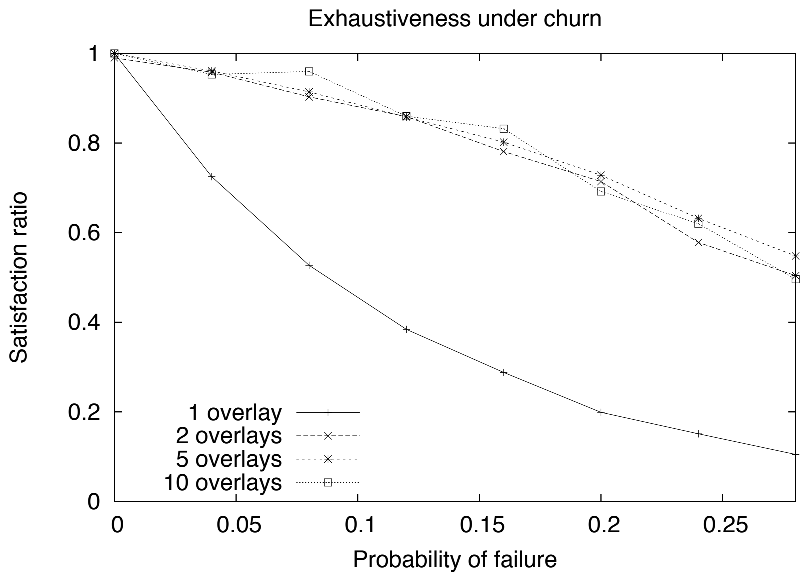


Figure 2.9: Exhaustiveness vs. churn rate

architectures are more robust and thus good candidates for information retrieval on dynamic platforms.

## 2.6 The Experimentations

### 2.6.1 JSynapse

In order to test our protocols on real platforms, we have initially developed JSynapse, a Java software prototype, which uses Java RMI standard for communications 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 software. We also provided some practical classes which help in automating the generation of the inter-overlay network and the testing of specific scenarios. We have experimented with JSynapse on the Grid'5000 platform [Cappello *et al.* 2005] connecting more than 20 clusters on 9 different sites. Again, Chord was used as the intra-overlay protocol.

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

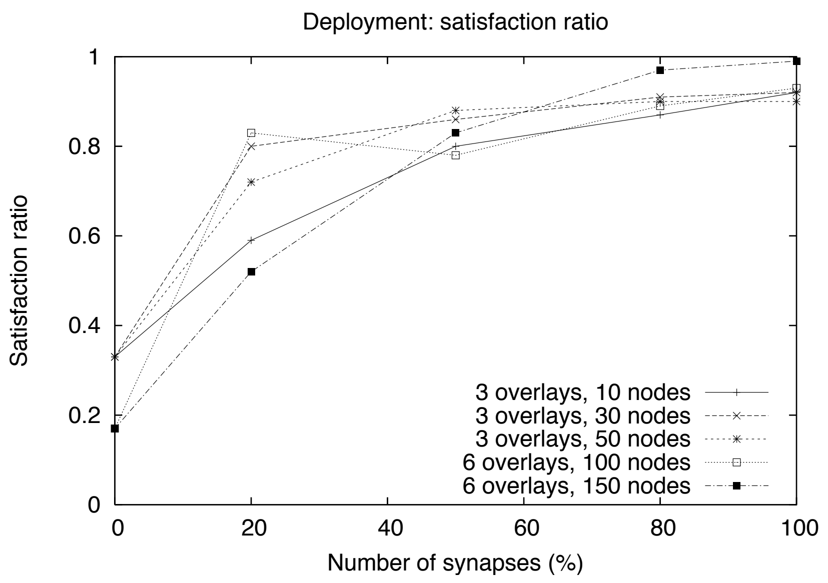


Figure 2.10: Deploying Synapse : Exhaustiveness

of which was always 2).

We give a proof of concept and show the viability of the Synapse approach while confirming results obtained by simulation. We also focus on the metrics affecting the user (satisfaction ratio and time to get a response). Once his request was sent, a user waits only 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 2.10 shows the satisfaction ratio when increasing the number of synapses (for both white and black box versions). 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 2.11 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.

### 2.6.2 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 `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 scenarii.

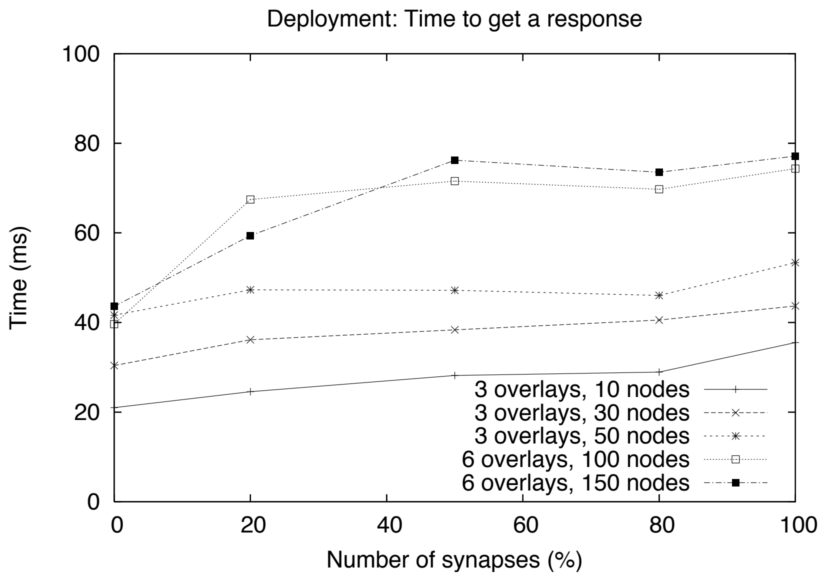


Figure 2.11: Deploying Synapse : Latency

## 2.7 Conclusion

In this chapter we have introduced Synapse, a scalable protocol for information retrieval in heterogeneous inter-connected overlay networks relying on co-located nodes and inter-routing policies of opportunistic nature. 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.

The contibutions of this chapter have been published as *Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks*, in Proceedings of NETWORKING 2010, Springer LNCS [Liquori 2010].

Chapter 3

# Optimal discovery mechanisms for distributed gateways

## Contents

In the previous chapter, we proposed a system to interconnect heterogeneous overlays by means of co-located nodes, i.e. nodes that can belong to several overlays at the same time, and act together as a form of distributed gateways. The system relied on what we called "opportunistic routing", i.e. a message could jump from one overlay to another only if it touched a gateway node during the routing in the first overlay. While proving how such an approach is scalable and allows for a quasi-exhaustive system with a low percentage of gateway nodes, the opportunistic routing is not the only available one, and not necessarily the optimal one for every application.

In this chapter, we present a novel protocol and software architecture that better formalises the capabilities of an interconnected system and allows for more flexible routing schemes. The protocol, named Synapse 2.0, exposes a set of API, independent from the underlying overlays, to exchange the required inter-routing information in an unstructured fashion, defines new optimized mechanisms for the discovery of the overlay gateways and offers a way to define new routing policies, including but not limited to the presented opportunistic routing. Furthermore, it generalises the previous approach to work in both collaborative and non collaborative scenarios.

## 3.1 Context

### 3.1.1 Evolving the Synapse protocol

The work on the *Synapse overlay interconnection framework* presented in Chapter 2 showed how, given an arbitrary and statistically uniform set of lookup requests for resources distributed across a set of structured overlay networks connected by nodes co-located into several overlays at the same time, it is possible to rely solely on the chance that a request would "touch" a gateway node (i.e. a *Synapse*) to perform inter-routing of a request into a foreign overlay and achieve quasi-exhaustivity with a relatively low number of well-connected gateways, as shown in Figure 2.8.

The intuition behind this result is that the consistent hash function used to encode resource keys and node identifiers in a structured overlay ultimately causes the messages to circulate evenly across the whole overlay, with the effect that, even with a low percentage on gateway nodes evenly distributed in the overlay, the chance for a request to encounter a co-located node along its routing path is fairly high.

Regardless of the effectiveness of such intuition, there still is a set of practical and functional challenges on the way to achieving a concrete meta-protocol suitable for real-world applications:

- Relying solely on an opportunistic mechanism works when the data stored in the DHT is assumed to be atomic, but may lose its effectiveness in case of complex data structures involving multiple indirections of keys pointing to other keys in the same overlays, as described in [Garces-Erice 2004][1]. In case of complex information, it becomes essential to be able to efficiently issue subsequent requests directed to the overlay where the first key was found, to avoid severe data consistency problems;

- Extending the routing to multiple overlays require the exchange of additional data, such as the non-hashed key (in the hypothesis that different overlays use different hash functions). In an opportunistic scenario, such data needs to be constantly carried around by the overlay messages, in the off chance that a gateway node would forward that message, thus increasing the overhead traffic in each overlay and breaking any backward compatibility with existing networks;

[1]Chapters 5 and 6 will present two cases where such complex encoding is required.

- A different mechanism needs to be put in place for those scenarios where backward compatibility has to be ensured, leading to inconsistencies between the "black-box" protocol (in scenarios when the overlay network protocol messages cannot be changed) and the "white-box" protocol;

- Furthermore, the black box implementation relies on the use of additional DHTs as the control network, to store the meta data that would have to be exchanged between peers during the inter-routing. That would add an additional layer of complexity in the implementation, and more delay when issuing a request, due to the fact that several DHT operations would have to happen before a message to be inter-routed can be reliably sent over the overlay;

- Due to the assumptions that different overlays may rely on different hash functions, and that hash functions should not be considered reversible, the need to exchange information deemed sensitive may arise. For example, to route a request in a foreign overlay a gateway node has to get the non-hashed key $nK$ from the node in the overlay issuing the request. In the protocol described in Chapter 2, $nK$ was either embedded in the overlay messages, or stored in the control network. Both choices pose problems related to both security ($nK$ is left visible to an arbitrary set of nodes routing the packets containing it) and performances (an application might decide to store information related to a file, using as a key the MD5 hash of the whole file.);

The above considerations drove the research presented in this chapter, where the Synapse protocol has been improved to fulfill the following goals:

- Enable the possibility to efficiently retrieve complex data structures across heterogeneous overlays, once the first key is found;

- Maintain as much as possible the same desirable performances in terms of exhaustivity in the overall system, by continuing to exploit, in a new way, the intuition described above;

- Reduce as much as possible the exposure of "sensitive" information, namely the un-hashed key $nK$, to the eyes of parties non-involved in the routing;

- Create a consistent behavior for the "black-box" and the "white-box" scenarios, i.e. develop a routing scheme consistent across the backward-compatible and non backward-compatible scenarios;

- Reduce the message overhead as much as possible, and increase the robustness of the meta-overlay created by the co-located nodes;

### 3.1.2 Towards a common architecture to interconnect overlay networks

With these premises in mind, the novel *Synapse 2.0* protocol implements the above points in the following way:

- Routing to a foreign overlay does not happen anymore opportunistically, but in a deterministic way, with the originating peer contacting directly a co-located node with a dedicated message that contains $nK$ and any other useful information (such as a TTL parameter, or a list of overlays to target). This communication can be easily be encrypted with a public/private key mechanism, and exposes the sensitive data only to the impacted parties (originating node and gateway);

- The use of a dedicated message exchange between originating node and gateway makes the mechanism compatible even with existing networks, where old nodes and new ones can coexist in the same network without the old nodes being impacted by any additional information being carried in the protocol packets;

- Gateways can be discovered using different mechanisms, depending on the "openness"of the system: in the best case, it can be achieved by simply adding an additional flag in the packets issued by a gateway node, and leaving every node forwarding the message reading said flag. This way, the same intuition that allowed for reaching quasi-exhaustivity during a message routing can be exploited to *discover* the gateways for later use;

- This mechanism leads to the creation of a control network in the form of an unstructured overlay, by having each node maintaining a constant list of pointers to gateway nodes being discovered. On top of this unstructured overlay, each node can independently adopt different routing schemes, depending on the need. A node could, for example, decide to organize its pointers in a structured way, based in the identifier of the networks they point to, thus mimicking a hierarchical overlay behavior;

## 3.2 The Synapse 2.0 Interconnection Framework

In this section, we describe in details the Synapse 2.0 protocol, the structure of a Synapse node, and the processes of inter-overlay routing and node discovery performed in Synapse. As a reference for operations and messages, we will adopt *KBR* routing, as described in the API of [Dabek ], since there an independent and implementation-agnostic abstraction is provided, which can be used for either structured, unstructured, or hybrid overlays.

### 3.2.1 Synapse Protocol Overview

As we have mentioned earlier, the idea behind Synapse is to provide a network framework to deal with transparent interconnection and collaboration of heterogeneous overlays. Throughout the rest of the work, we will refer to three types of nodes: *legacy-nodes*, *synapse-nodes*, and *gateway-nodes*:

- Legacy-nodes are simple instances of one single overlay protocol, they are connected to just one overlay, and are unaware of the Synapse protocol;
- Synapse-nodes are nodes which are aware of the Synapse protocol. They can be connected to one or more overlays (and, as such, they are also able to route messages in each of these overlays). Each synapse-node maintains a Direct Overlay Table (DOT) with pointers to other gateway-nodes it is aware of. The main functionalities of a synapse-node will be described later on in this section;
- Gateway-nodes are a special case of synapse-nodes, and are necessarily connected to two or more overlays.

Additionally, for each synapse-node, we will be referring to three sets of overlays related to it: *connected-overlays*, *direct-overlays*, and *indirect-overlays*:

- An overlay $o$ is a connected-overlay of a given synapse-node $s$ iff $s$ is connected to $o$;
- An overlay $o$ is a direct-overlay of a synapse-node $s$ iff there exists a gateway-node $s'$ in the DOT of $s$, such that $o$ is a connected-overlay of $s'$;
- An overlay $o$ is an indirect-overlay of a synapse-node $s$ iff it is neither a connected-overlay of $s$, nor a direct-overlay of $s$.

Each overlay network is identified by a unique `netID`. Figure 3.1 shows the topology of five different overlay networks having ten synapse-nodes. Here, for instance, the connected-overlays of the synapse-node `S1` are the overlays `netID1` and `netID5`, while its direct-overlays are `netID2`, `netID3`, and `netID4` (via gateway-nodes `S2`, `S5`, and `S8`).

**Hashing.** Without loss of generality, we assume that all of the overlays expose key-based routing capabilities. Also, we do not require that all of the overlays use one hash-function, or are aware of all of the hashing functions. As un-hashing is not possible, routing a message outside of an overlay involves the exchange of the non-hashed key.

### 3.2.2 Synapse-node functionalities

A synapse-node must be able to:

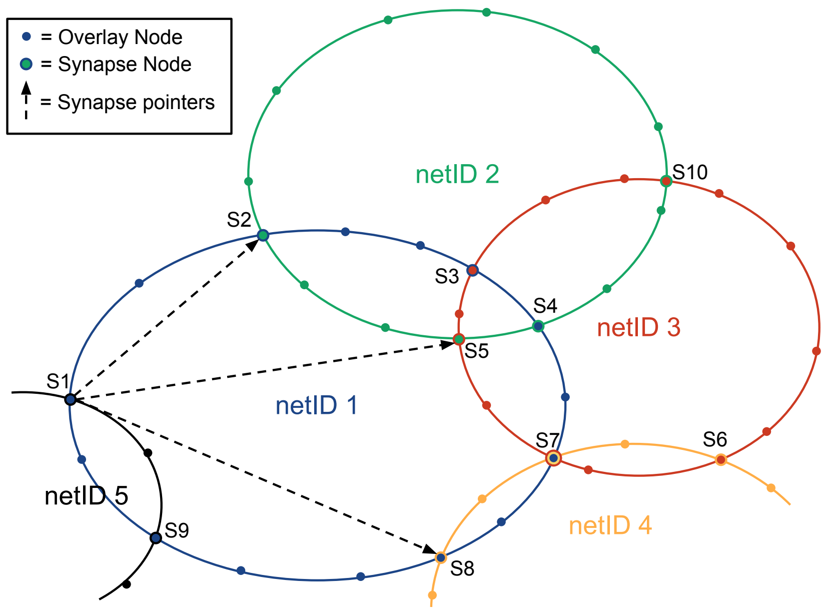


Figure 3.1: Overlays with gateway-nodes

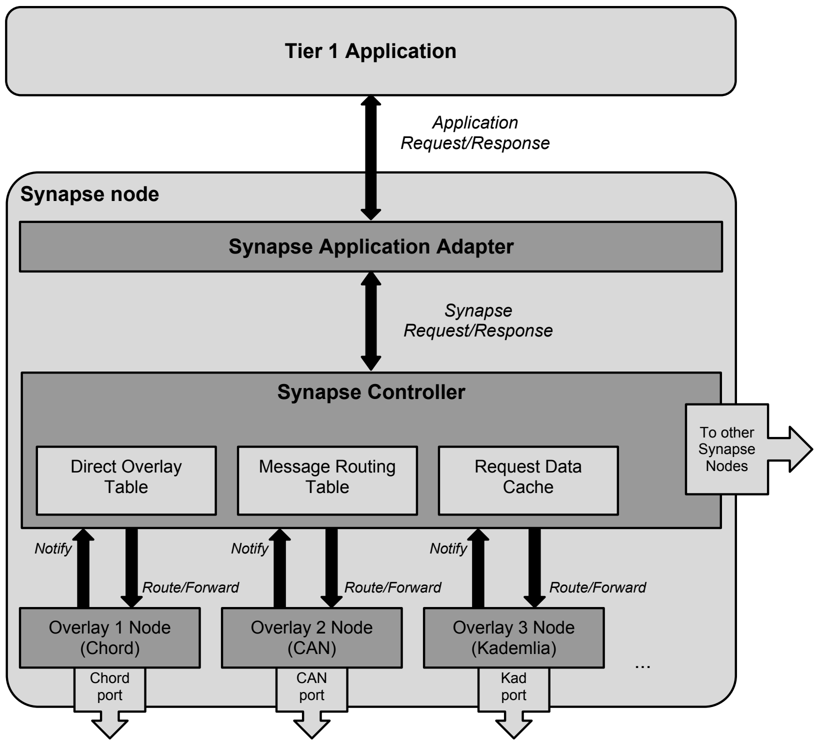


Figure 3.2: Structure of a synapse-node

- Process and route a message from the application layer to its connected-overlays;
- Dispatch a message directly to other synapse-nodes, to be routed in overlays others than the connected ones;
- Routing a request coming from other synapse-nodes;
- Discovering new synapse-nodes;
- Inviting new synapse-nodes to join its connected-overlays.

Apart from the data structures and messages which need to be maintained for each of the connected-overlays (finger tables, neighbors lists, routing messages, etc.), a synapse-node must handle new data structures so as to deal with inter-overlay routing:

- a *Network Identifier (netID)* per each overlay, to identify it unequivocally;
- a *Direct Overlay Table (DOT)*, which is a table in which pointers to gateway-nodes are stored, arranged per netID of the overlays they are connected to;
- a *Message Routing Table (MRT)*, responsible for storing information about ongoing messages (TTL, source nodes, RequestID, targeted overlays, etc.);
- a *Cache Table (CHT)*, used for storing values which are associated with frequently requested keys, in order to minimize routing for popular items.

Furthermore, each synapse-node should be able to interpret the following messages:

- `SYNAPSE_OFFER(netIDList)`, issued by a gateway-node in order to publish the list of overlays it is connected to;
- `SYNAPSE_REQUEST(nonHashedKey,...)`, sent by a synapse-node to a gateway-node in order to route a message outside of an overlay;
- `SYNAPSE_RESPONSE(...)`, used by a gateway-node to return response messages for a `SYNAPSE_REQUEST`;
- `SYNAPSE_INVITE(netID)`, sent to a synapse-node to "invite" it to join a specific overlay;
- `SYNAPSE_JOIN(netID)`, issued by a synapse-node wishing to join a given overlay.

### 3.2.3 Synapse Routing Protocol

Message routing can use three different mechanisms in synapse-nodes:

- they can route a message to any of their connected-overlays;
- they can also route a message to any of their direct-overlays by issuing a `SYNAPSE_REQUEST` message to a gateway-node in its DOT;
- they can also reach their indirect-overlays by issuing a `SYNAPSE_REQUEST` message, with the target netIDs specified, to a random set of gateway-nodes. This starts an unstructured routing mechanism through gateway-nodes until a node connected to the target overlay is found.

A `SYNAPSE_REQUEST` message carries several parameters, amongst which are the non-hashed key for the message, a RequestID (in order to identify if a message has already passed through a gateway), a TTL parameter which defines how many times should a message be routed to subsequent gateways and, if necessary, a list of target netIDs to which the message should be routed specifically. The choice of which, and how many overlays to select for message routing constitutes the routing strategy of the system.

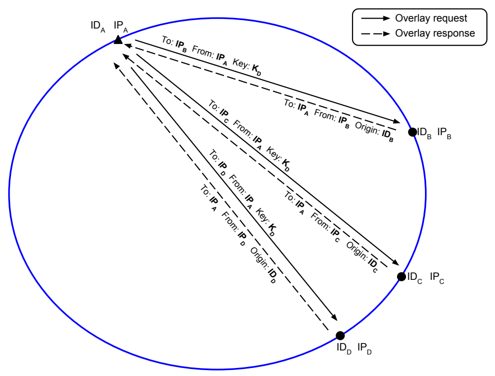
### 3.2.4 Gateway node discovery strategies

In order to reach direct overlays, a synapse-node which joins the network needs to discover gateway-nodes connected to overlays other than his. There are several mechanisms at hand, depending on the application scenarios:

- *Message embedding (Passive Discovery):* in a collaborative scenario, in which the overlay protocol messages can support additional data, the simplest solution is to embed into the message 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 DOT, in Kademlia-like fashion;
- *Active notifications (Active Discovery):* being notified of a transiting message, a synapse-node can decide to proactively send a `SYNAPSE_OFFER` message, containing the list of its connected overlays, to the source node, 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);
- *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 synapse-nodes via an iterative peer exchange mechanism. This, however, requires an initial synapse-bootstrap-node to be contacted, in order to perform the first discovery;

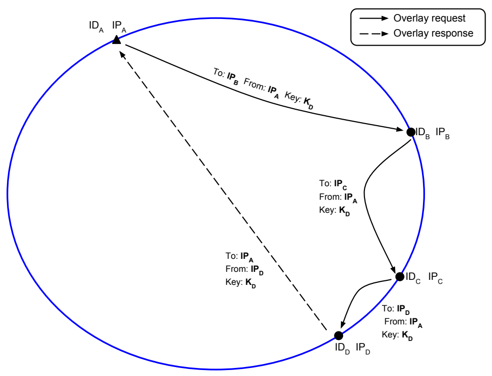
- *Aggressive discovery:* apart from these strategies, which are generic and suitable for any overlay protocol, other strategies can be put in place within a collaborative scenario, to exploit specificities of a certain protocol (e.g. a source node list, leaf tables, neighbor cache, etc.).

**A taxonomy of collaboration scenarios.** The choice of a peer discovery mechanism over the others strictly depends on the capabilities of the supporting overlay networks, as well as the scenario in consideration.



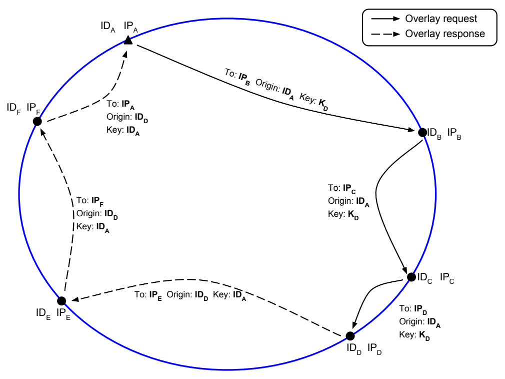
Overlay request
Overlay response
$ID_A$ $IP_A$
$ID_B$ $IP_B$
$ID_C$ $IP_C$
$ID_D$ $IP_D$
To: $IP_B$ From: $IP_A$ Key: $K_D$
To: $IP_A$ From: $IP_B$ Origin: $ID_B$
To: $IP_C$ From: $IP_A$ Key: $K_D$
To: $IP_A$ From: $IP_C$ Origin: $ID_C$
To: $IP_D$ From: $IP_A$ Key: $K_D$
To: $IP_A$ From: $IP_D$ Origin: $ID_D$

(a) Iterative routing



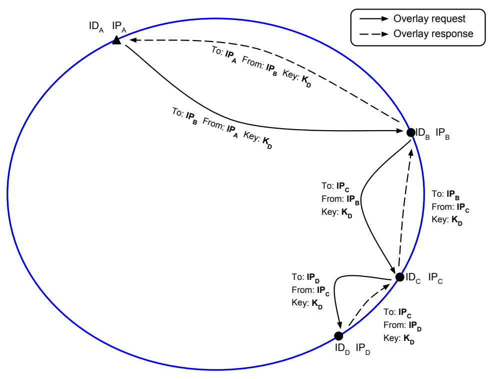
Overlay request
Overlay response
$ID_A$ $IP_A$
To: $IP_B$ From: $IP_A$ Key: $K_D$
$ID_B$ $IP_B$
To: $IP_C$ From: $IP_A$ Key: $K_D$
To: $IP_A$ From: $IP_D$ Key: $K_D$
$ID_C$ $IP_C$
To: $IP_D$ From: $IP_A$ Key: $K_D$
$ID_D$ $IP_D$

(b) Semi-recursive routing



Overlay request
Overlay response
$ID_A$ $IP_A$
To: $IP_B$ Origin: $ID_A$ Key: $K_D$
$ID_B$ $IP_B$
$ID_F$ $IP_F$
To: $IP_A$ Origin: $ID_D$ Key: $ID_A$
To: $IP_F$ Origin: $ID_D$ Key: $ID_A$
To: $IP_C$ Origin: $ID_A$ Key: $K_D$
$ID_C$ $IP_C$
$ID_E$ $IP_E$
To: $IP_E$ Origin: $ID_D$ Key: $ID_A$
To: $IP_D$ Origin: $ID_A$ Key: $K_D$
$ID_D$ $IP_D$

(c) Full-recursive routing



Overlay request
Overlay response
$ID_A$ $IP_A$
To: $IP_A$ From: $IP_B$ Key: $K_D$
To: $IP_B$ From: $IP_A$ Key: $K_D$
$ID_B$ $IP_B$
To: $IP_C$ From: $IP_B$ Key: $K_D$
To: $IP_B$ From: $IP_C$ Key: $K_D$
To: $IP_D$ From: $IP_C$ Key: $K_D$
$ID_C$ $IP_C$
To: $IP_C$ From: $IP_D$ Key: $K_D$
$ID_D$ $IP_D$

(d) Source-recursive routing

Figure 3.3: Examples of different overlay routing mechanisms

Concerning the routing type adopted by the underlying overlays, we can summarily group it in the followinfg classes:

1. **Iterative routing (Figure 3.3(a)):** the originating node of a message takes the routing exclusively upon itself. It directly contact nodes at every step of the routing path, and receives as a response the subsequent hops to contact. In this scenario every message circulating in an overlay must contain the contact endpoint (i.e. its IP address) of the originating node.

2. **Semi-recursive routing (Figure 3.3(b)):** here a request message is routed recursively form one hop to the subsequent, until the destination is found. However, responses can be sent directly to the originating node, and therefore the originating endpoint is carried in every message.

3. **Full-recursive routing (Figure 3.3(c)):** a request follows the same path in the semi-recursive. However only the overaly identifier of the originator is knwon and any response must be routed back folliwng the overlay routing rather than a direct path.

4. **Source-recursive routing (Figure 3.3(d)):** this routing also happens recursively, however every hop only knows the previous one, and the originator is known only by the first hop. To be routed back, a response message has to follow the request routing path backwards.

Another important distinction lies in the cooperation scenario taken into consideration upon the design of the system, in particular:

- When maintaining a backward compatibility with existing systems is not an issue, we can talk about **collaborative scenarios**. It can be the case when designing a new system from scratch and not having to interact with existing networks with nodes laready deployed. The interest in this comes form the fact that, in such scenarios, it is possible to alter the overlay messages to piggyback additional information (e.g. the list of connected overlays for each node in the network)

- If, however, backward compatibility is at stake, we talk about **non-collaborative scenarios**. As a trivial example, think of the possibility of connecting a new set of overlays to the existing Kademlia networks used in Bittorrent. In such scenarios, we are limited in the possibility of altering the underlying protocol messages, due to the presence of legacy nodes who might not be able to undestand the additional information carried.

Depending on the routing type of the underlying overlay networks, we can choose to adopt one of the aforementioned peer discovery techniques, as summarized by Table 3.1.

| | **Collaborative** | **Non-collaborative** |
|---|---|---|
| **Iterative** | Message embedding | Active notifications |
| **Semi-recursive** | | Active notifications |
| **Full-recursive** | | Peer exchange |
| **Source-recursive** | | Active notifications, Peer exchange |

Table 3.1: Best gateway discovery technique per routing type and collaboration scenario.

The table shows what can be considered the optimal peer discovery in terms of maintenance messages overhead. As one can see, every collaborative scenario can rely on the messagge embedding, which best ensures that the information related to a new gateway node will be discovereb by the higest number of nodes. In non-collaborative scenarios, however, nodes can arely on Active notifications by the gateway nodes themselves in all those scenarios where a gateway node can retrieve the IP address of the node to contact. When this is not possible, nodes can still rely on a classical peer exchange mechanism to refresh their routing tables.

### 3.2.5 Synapse-node routing strategies

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

- *n-Random routing:* a synapse-node picks $n$ random overlays to which to route the request, out of all of its connected and direct overlay;
- *n-Flood routing:* a synapse-node picks $n$ nodes per *each* direct and connected overlay. The choice of replicating a message onto the same overlay stems from the need to overcome network partitioning by routing a request through nodes placed in different locations of the addressing space;
- *n-Direct routing:* a synapse-node routes a message directly, and only to a certain overlay, by picking $n$ synapse-nodes connected to said overlay. If no finger to this overlay is present, the message can be routed to random synapse-nodes by sending a `SYNAPSE_REQUEST` message with the list of target networks specified;
- *Opportunistic routing:* a synapse-node can dispatch a `SYNAPSE_REQUEST` to another synapse-node upon receipt of a `SYNAPSE_OFFER`, thus having a much higher chance of routing to an active node.

### 3.2.6 Synapse-node structure.

As shown in Figure 3.2, a synapse-node consists of several components:

- The *Synapse-controller* is responsible for orchestrating multiple requests, routing messages according to the appropriate strategy, and collecting and grouping results arriving from different overlays. It also takes care of the maintenance of the synapse overlay, by performing discovery of new synapse-nodes, checking their state via ping messages, and dispatching join invitations. It maintains and relies on the DOT to store pointers to gateway-nodes and on the MRT to keep track of ongoing routings. Furthermore, the CHT can store recently retrieved values, in order to be able 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 RPC call is received, in order to check for new gateway-nodes, by examining whether or not a netID list is present in the message header, or to announce its own presence via a `SYNAPSE_OFFER` message;

- The *Synapse-application-adapter* acts as an interface to and from the application layer. It serves to decouple the applicative part from the background multi-overlay logic, by exposing an API agnostic of the underlying structure, processing complex queries, and to generate appropriate messages for the synapse-controller.

### 3.2.7 Self-organization via "social networking" primitives.

In addition to Synapse messages, we propose a set of primitives which would serve to implement overlay self-organization mechanisms. By issuing a `SYNAPSE_INVITE` message, a synapse-node can propose to other synapse-nodes that they join one or more overlays, in order to, for instance, increase the overlay capacity, QoS, or external connectivity. In a similar manner, a synapse-node can offer to become 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 could be particularly interesting to consider in a scenario where an overlay would be able to "shrink" or "grow" around application data, such as, for instance, the social graph in online social networks. They can also be exploited to regulate connectivity of an overlay towards the rest of the system, by increasing the number of gateway-nodes to overlays in question, providing a flexible mechanism to implement QoS and failure avoidance in a system.

## 3.3 A routing example

We hereby present an example of routing in a Synapse network, using a Random Walk strategy with opportunistic routing enabled. Figure 3.4 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 and the Synapse Application Adapter, translates it into a `MULTI_GET(K1, strategy=RANDOM.1.1)` message to the Synapse Controller, the strategy to adopt.

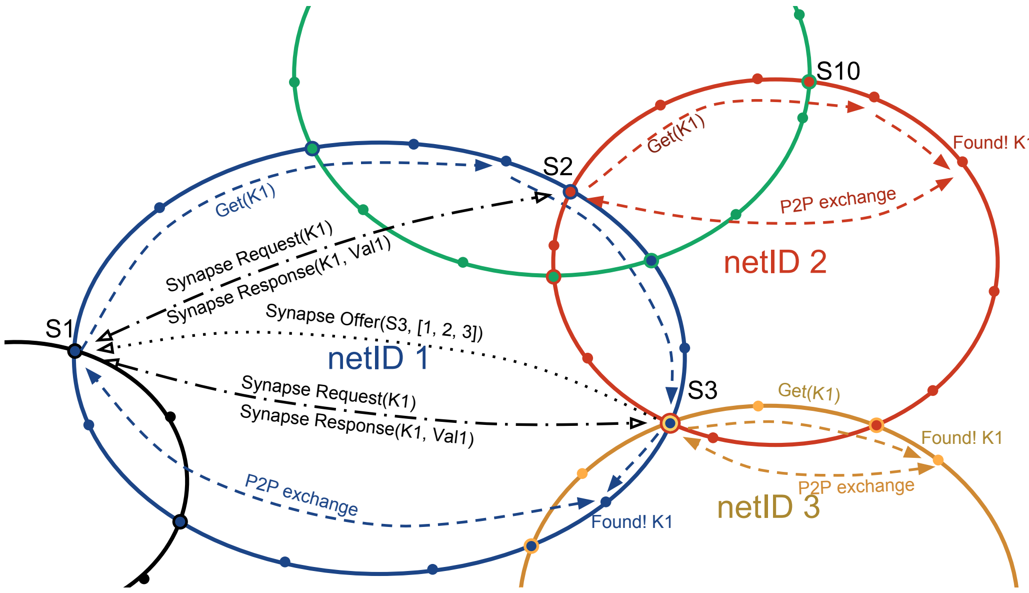


Figure 3.4: Routing in Synapse

2. The Synapse Controller, according to the 1-Random-Walk strategy, picks 1 random overlay (*netID1*) from the connected overlay list and 1 random node (*S2*) from the Direct Overlay Table.

3. It routes directly a `GET(Hash(K1))` message in *netID1* and, in parallel, `SYNAPSE_REQUEST(K1, RequestID, TTL=1, strategy=RANDOM.1.1, visited=[netID1], S1PubKey)` to *S2*.

4. *S2*, upon reception of the `SYNAPSE_REQUEST`, picks 1 random connected overlay (*netID2*) to reroute the request and decreases the TTL value. Since now TTL=0, the request is not routed any further to other gateway nodes.

5. During the routing in *netID1*, another gateway node (*S3*) in netID1 forwards it. *S3*, first updates its DOT with *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)`.

6. *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)`.

7. *S3*, receiving the `SYNAPSE_REQUEST`, picks overlay *netID3*. and routes a `GET(Hash(K1))`.

8. Eventually, the requests in *netID1*, *netID2* and *netID3* will reach its destination nodes, and responses will be sent back to *S1*, *S2* and *S3*.

9. *S2* and *S3* will send the message response `RESP1` back to *S1* via a `SYNAPSE_RESPONSE(ReqID, RESP1, O2)` message, encrypted with *S1*'s public key.

10. 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*.

## 3.4 Protocol implementation in OverSim

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

- 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 [Dabek ].

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. 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.
**Extending OverSim's overlay host.** Figure 3.5 shows the Synapse Controller

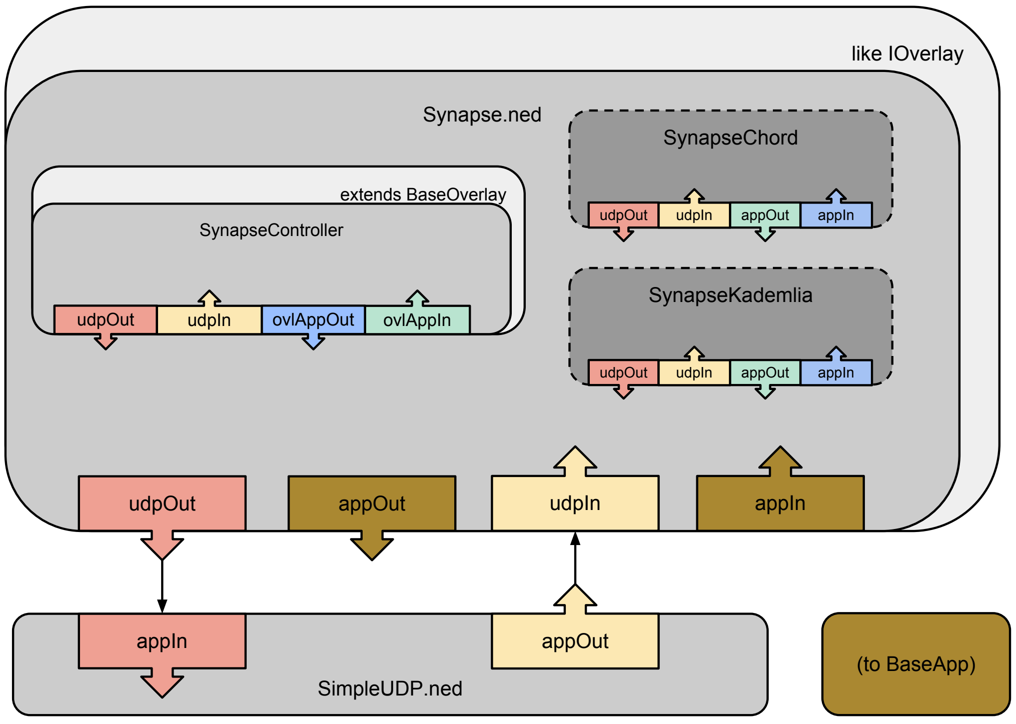


Figure 3.5: Synapse OverSim modules diagram

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 a 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 could join a new overlay at runtime.

**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 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:

```
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 [syn ].

## 3.5 Simulation Results

In this section, we present some results obtained by running simulations within the OverSim-based Synapse simulator. In the simulations, all of the nodes were treated as synapse-nodes, and some were treated as gateway-nodes.

### 3.5.1 Simulation settings

Simulations were run on 2000 nodes, clustered into sub-overlays, half of which were Chord, and half of which Kademlia. All of the nodes were treated as synapse-nodes, i.e. all can perform active and passive discovery of gateway-nodes, and no legacy nodes are present.

With the system being composed of nodes clustered across many different overlays, which are connected with one another through unstructured routing, the main purpose of our simulations has been to test the reachability of each of the overlays, while varying the granularity of the network (i.e. the number of different overlays, given the same overall amount of nodes), the number of gateway-nodes present in each of the overlays and the average connection degree of gateway-nodes (i.e. the number of different overlays a gateway-node is connected to).

The tests have involved inserting random keys throughout the entire system and performing lookups for said keys, by a different node, which is not necessarily a member of the same sub-overlay in which the key is present. All replication within the sub-overlays has been disabled in order to create the most challenging conditions, and produce metrics as correlated as possible.

As the idea was to gather a lower bound of the performances of the system, we have chosen to adopt the simplest and least demanding routing strategy for synapse requests: a stateless 1-Random-Walk, meaning that nodes, at each of the routing steps, would route to every connected overlay but choose only 1 random gateway-node amongst all to reroute the request to, without considering past routing steps. Finally, the TTL has been set to 8 for all of the simulations.

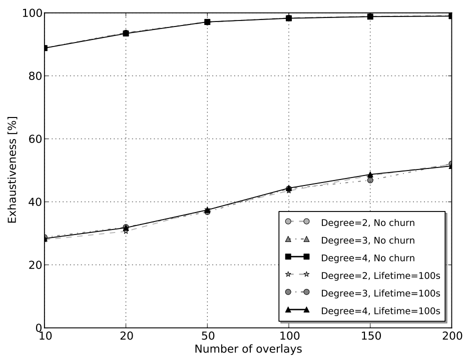
We tested different scenarios, without churn, to evaluate the topology built by the node discovery process, and with high churn, i.e. with a very short node lifetime, to test it in extreme conditions, such as those of a mobile application. In all of the simulations, the connection degree was equal for all gateway-nodes. However, the percentage of gateway-nodes and their interconnection degree have been correlated to guarantee the minimum number of gateway-nodes-per-overlay to have a connected topology across all sub-overlays, without leaving any sub-overlay isolated due to, possibly, a lack of gateway-nodes connected to it. The relationship between the connection degree and the number of gateway-nodes is explained in the following subsection.

**Mathematical Background** Let us denote by $s$ the number of synapse-nodes, by $g$ the number of gateway-nodes, by $d$ the degree of connectivity of the gateway-nodes, by $o$ the number of overlays, and by $n$ the overall number of nodes, calculated as $n = s + g$. Apart from these, we will require two “extended” notions: the extended number of gateway-nodes, $g_e = d \cdot g$, and the extended overall number of nodes $n_e = s + g_e = s + d \cdot s$. Using this, we can calculates:

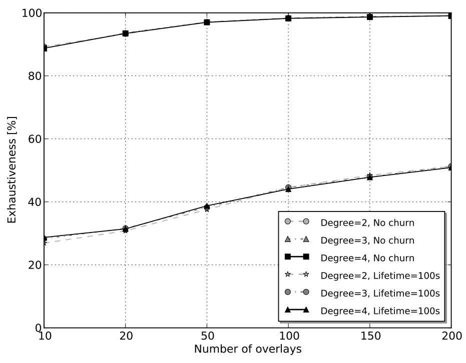
- the number of nodes-per-overlay, $n_o = \frac{n_e}{o} = \frac{l+d \cdot g}{o}$;
- the number of gateway-nodes-per-overlay, $g_o = \frac{g_e}{o} = \frac{d \cdot g}{o}$;
- the overall percentage of gateway-nodes, $s_{\%n} = \frac{g}{n}$;
- the percentage of gateway-nodes-per-overlay, $s_{\%o} = \frac{g_o}{n_o} = \frac{g_e}{n_e} = \frac{d \cdot g}{l+d \cdot g}$.

### 3.5.2 Topology construction

Topologies have been created statically, using $n$, $o$, $d$, and, depending on the simulation scenario, either the percentage of gateway-nodes-per-overlay, or the overall



(a) FIT Topology



(b) RAT Topology

Figure 3.6: Effects of system granularity, with and without churn

percentage of gateway-nodes in the system. Two algorithms were used to generate topologies:

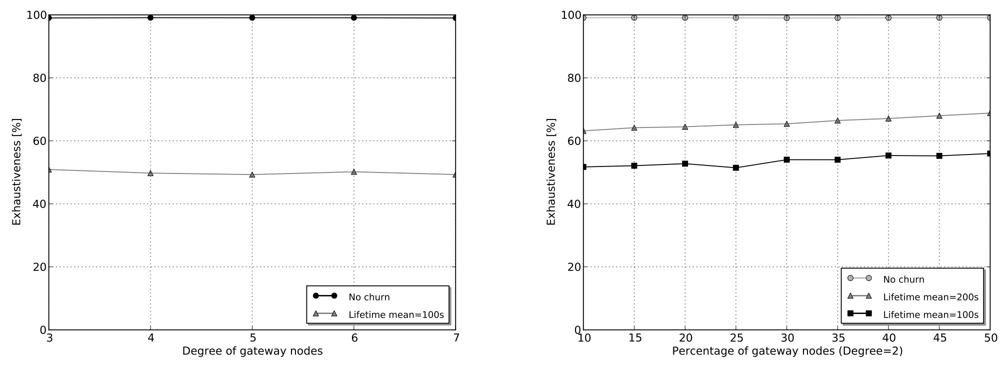
1. FIT – the topology is constructed to be fully-interconnected, in the sense that from any overlay there exists a path through gateway-nodes of the system to any other overlay. This requires at least $\lceil \frac{o-1}{d-1} \rceil$ gateway nodes to be present in the system, and is accomplished using an algorithm described in the web appendix;

2. RAT – The topology is constructed with fully random assignments of overlays to gateway-nodes, using a uniform distribution over the $o$ overlays.

Figure 3.6 shows the effect that system granularity (i.e. the number of sub-overlays) has on the general system exhaustiveness. We have simulated both a churn-less environment and one with high churn, to test the topology itself, as well as its resilience to extreme conditions. Figures 3.6(a) and 3.6(b) compare a completely random topology vs. one where exhaustive connectivity has been forced. It is remarkable that the performances are substantially equivalent, suggesting that, in fact, the gateway topology can generally be built with just a partial knowledge of the system by a simple random selection of overlays. Even with 200 overlays, the routing has proved to be exhaustive, reaching every sub-overlay, and suggesting that building a clustered overlay network is a feasible solution. The lower exhaustiveness with lower granularity is explained by the fact that, having a higher number of edges for each overlay, loops can be present, leading, with this simplest routing strategy, to requests bouncing back to the overlay they came from, an effect that can easily be avoided with a stateful routing strategy.

### 3.5.3 Configuration of gateway-nodes

Since maintaining a connection to multiple overlays is a costly operation, in this experiment we have tested the effectiveness of two opposite scenarios, one with



(a) Few high-degree gateways

(b) Many low-degree gateways

Figure 3.7: Performance comparison for different gateway topologies

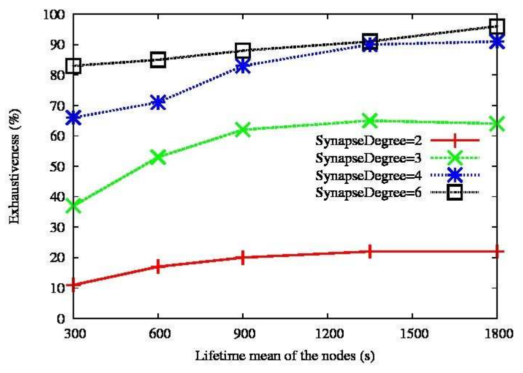
very few nodes maintaining a high degree of connectivity (much like a super-peer structure), and a second one, in which an increasing number of nodes maintains a connectivity as low as possible (degree 2). It is worth noting that, despite the high connectivity degree, the gateway nodes in the first scenario were not exempted from churning.

Figure 3.7 shows the performances in the two scenarios. Interestingly enough, a decrease from degree 6 to degree 3 (Figure 3.7(a)) does not bring any visible decrease in performances, neither with or without churn, due partly to the simple routing strategy adopted, and it is an aspect that can be taken into account when designing a system by explicitly deploying synaps-gateways. In the second scenario (Figure 3.7(b)), on the other hand, the increase of gateway-nodes brings a slight increase in the exhaustiveness under churn, which suggests a possible strategy to handle situations of sudden churn in a system, by having most of the nodes immediately increase their connectivity degree by 1.

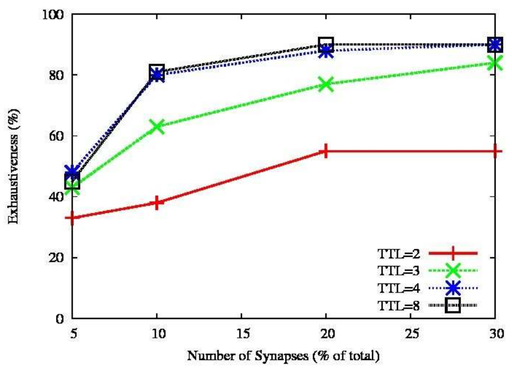
## 3.6 Experimental Results

In order to evaluate the behavior of Synapse within a real-world environment, we have developed a Java implementation of the Synapse protocol, which we have used to perform experiments on the national French Grid'5000 platform, that aims at providing an experimentation testbed to study large scale parallel or distributed systems which comprises thousands of interconnected computers across numerous sites in France. In all of the experiments performed, we have used 1000 nodes, distributed over 10 Chord and 10 Kademlia overlays, interconnected via the Synapse protocol.

In the first experiment, we have investigated the exhaustiveness of the interconnected systems under different mean lifetimes of the nodes and different degrees of connectivity of synapses-nodes. We have placed an emphasis on high-churn-rate



(a)



(b)

Figure 3.8: Experiments on Grid5000

conditions (when the mean lifetime of the nodes is low), which should be observable in the near future, in overlay networks in which peers need not only be desktops and laptops, but also Internet TV and mobile devices, which are expected to join and leave the network at high frequency. In order to generate this high churn rate of nodes in the systems, we have used the Pareto distribution. The experiment was performed for mean lifetime values between 300s and 1800s, with the degree of connectivity of the synapses-nodes varying between 2 and 6, once for each of the combinations. The overall percentage of synapses-nodes was fixed to 20% of the overall number of nodes, while the TTL value was fixed to 8, in all of the cases. The results obtained from this experiment are shown in Figure 3.8(a).

As can be seen from Figure 3.8(a), for a fixed degree of connectivity, the Synapse protocol is fairly resilient for values of the mean lifetime above 900s, and less resilient for lower values. However, in order to achieve a sufficient level of exhaustiveness, it is necessary to increase the degree of connectivity of synapses-nodes to at least 4, for mean lifetime values above 900s, or to at least 6, for mean lifetime values below 600.

In the second experiment, we have once again investigated the exhaustiveness of the interconnected systems, this time while varying the percentage of synapses-nodes from 5% to 30%, and the TTL from 2 to 8, once for each of the combinations. The degree of connectivity of synapses-nodes has been fixed to 4, and the churn rate of the nodes to 1800s. The results obtained from this experiment are shown in Figure 3.8(b). It can be noticed from Figure 3.8(b) that the exhaustiveness significantly increases when the TTL is increased from 2 to 4, but remains the same as the TTL is increased from 4 to 8, giving rise to the conclusion that a TTL of 4 is efficient enough when interconnected networks of this scale are concerned (20 networks, 1000 nodes overall)[2]. One other inference which can be made from Figure 3.8(b) is that

[2]Given this result, one might question our choice of TTL in the first experiment. The reason for it being set to 8 there is the simple fact that the first experiment was performed prior to the second.

having 20% of overall nodes to be synapses will result in sufficient exhaustiveness for this scale of interconnected overlays, as there is an obvious rise in exhaustiveness accompanying the increase of the number of synapses-nodes from 5% to 10% and from 10% to 20%, while no further significant rise occurs with further increase of the number of synapses from 20% to 30%.

## 3.7 Conclusion

In this chapter we have extended and generalized the Synapse protocol presented in Chapter 2. This new version exposes a common set of APIs and a unified mechanism to deal with collaborative or non-collaborative scenarios, and allows for the implementation of more flexible routing schemes.

The protocol has been developed and evaluated in the OverSim overlay simulator, which has been modified to support multiple overlay types at run-time, and a Java client has been deployed and tested on the Grid'5000 platform.

The contributions of this chapter have been published as
*Towards a Common Architecture to Interconnect Heterogeneous Overlay Networks*, in Proceedings of the HotPOST Workshop, 17th International Conference on Parallel and Distributed Systems (ICPADS), 2011, IEEE [Ciancaglini 2012b]
and
*An Extension and Cooperation Mechanism for Heterogeneous Overlay Networks*, in Proceeding of the HetNETS Workshop, NETWORKING Conference, 2012, LNCS [Ciancaglini 2012c].

Chapter 4

# Modeling of interconnected systems

## Contents

The interconnection of overlay networks presents the biggest advantages when the number of nodes is substantial. In such scenarios, however, it becomes difficult to evaluate, by means of simulation or deployment, the performance of an interconnected system and estimate the optimal parameters for an arbitrary system. To overcome this strong limitation, we develop a generalized random graph based model to represent the topology of one unstructured P2P network, the partition of nodes into Synapses, the probabilistic flooding based search algorithms, and the resource popularity.

By knowing the structure of the overlay networks, in terms of neighborhood distribution, and the configuration of gateway nodes, we are able to evaluate the probability of finding a resource of known popularity and the average number of messages to reach it. We validate the model by means of a validation software written form scratch that heavily exploits concurrent programming techniques to be able and simulate simple networks in the order of millions of nodes, proving that its predictions are reliable and accurate. We use the model to investigate the performance and the cost of different search strategies in terms of the probability of successfully locating at least one copy of the resource and the number of queries as well as the interconnection cost. We also gain interesting insights on the dependency between interconnection cost and statistical properties of the distribution of Synapses.

Furthermore, we present some examples on how the modeling can be exploited by a network designer to determine, given initial conditions, the optimal configuration of an interconnected system, in terms of percentage of gateway nodes and average connectivity of each node.

## 4.1 System description

## 4.2 System description

In this chapter, we focus on unstructured P2P networks where peers organize into an overlay network by establishing application level connections among them. The topological properties of an overlay network are represented by the number of connections of any of its participants. To this end, we describe an overlay by means of the degree distribution $\{p_k\}$ that can be interpreted as the probability that a randomly chosen peer has $k$ connections in the overlay ($\sum_{k=1}^{\infty} p_k = 1$).

We consider a set of $X$ unstructured P2P networks that are interconnected thanks to a subset of peers that belong to multiple overlays (these special peers are denoted as *Synapses*). Any peer may then belong to $i \in \{1, \dots, X\}$ overlays: we denote $i$ as the *Synapse degree* of a peer. The interconnected system is then described by $\{s_i\}$ ($i \in \{1, \dots, X\}$) where $s_i$ is the fraction of peers belonging to $i$ overlays ($\sum_{i=1}^{X} s_i = 1$).

The search algorithm we consider is *flooding-based*. A peer starting a search sends *queries* to a randomly chosen subset of its one-hop neighbors. These nodes forward the queries to a randomly chosen subset of their one-hop neighbors, excluding the query originator, and so on until the maximum number of allowed hops, i.e. the query time-to-live (TTL). We also consider a variation of this search algorithm where a query is not forwarded by peers that own a copy of the resource. We focus on probabilistic versions of this general algorithm where any peer flips a coin before sending or forwarding a query to a specific neighbor. We allow the weight of this coin to be dependent on the Synapse degree of a peer; hence, a peer that belongs to $i$ overlays sends/forwards a query to a particular neighbor with probability $p_f(i)$ ($i \in \{1, \dots, X\}$)[1].

The goal of a search is to localize at least one resource related to the key we are looking for. There could be more replicas of the same resource hosted by different peers for two reasons: a resource is popular and/or is owned by peers located in different P2P networks. We represent resource popularity by $0 \leq \alpha \leq 1$, the average fraction of nodes that globally hold a copy of a given resource, and interpret it as the probability that a randomly chosen peer owns a copy of the resource.

[1]Please note that $\{p_f(i)\}$ ($i \in \{1, \dots, X\}$) *is not* a probability distribution hence in general $\sum_{i=1}^{X} p_f(i) \neq 1$.

All of the notation is summarized in Table 4.1 and a simple schema of interconnection through synapses is depicted in Figure 4.1.

Table 4.1: Chapter notation.

| Parameter | Description |
|---|---|
| $X$ | Number of interconnected P2P networks. |
| $p_k$ | Fraction of peers with $k$ connections in an overlay. |
| $s_i$ | Fraction of peers belonging to $i$ overlays. |
| $p_f(i)$ | Probability to send/forward a query to a neighbor for peers that belong to $i$ overlays. |
| $\alpha$ | Average fraction of nodes owning a copy of a resource |
| TTL | Query time-to-live. |

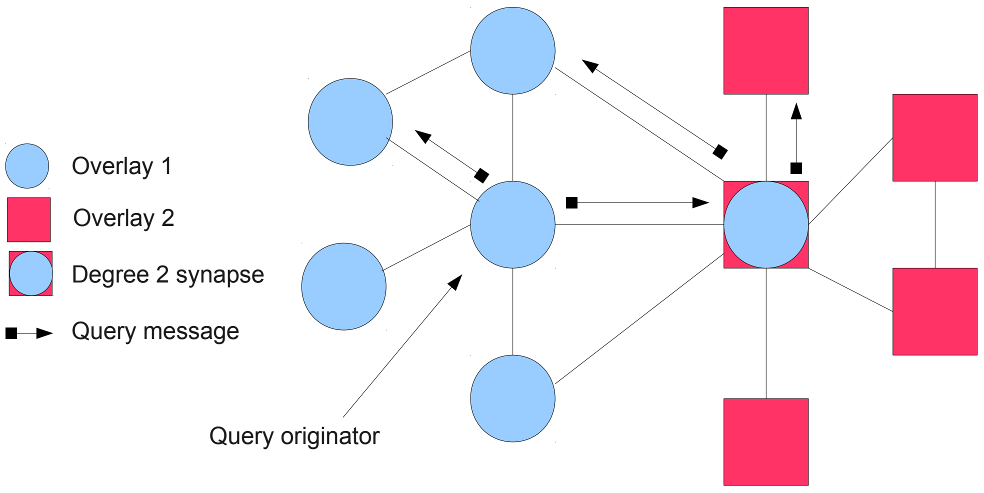


Figure 4.1: Example of two P2P interconnected networks ($X = 2$) and one degree 2 synapse that belongs to both.

## 4.3 System model

This section illustrates the random graph modeling approach to represent one overlay topology, the interconnection of $X$ P2P networks, the search algorithm, and resource popularity as described in Section 4.1.

### 4.3.1 One overlay topology

Each P2P network is organized into an overlay that we model as a generalized random graph whose degree distribution is $\{p_k\}$ that can be interpreted as the probability that a randomly chosen peer has $k$ connections in the overlay. The random graph degree distribution is a probability distribution therefore we consider its probability generating function (henceforth denoted as p.g.f.) that is equal to

$$G_0(z) = \sum_{k=0}^{\infty} p_k \, z^k \tag{4.1}$$

To correctly characterize the neighborhood of a randomly chosen peer we also need to characterize the probability distribution of the number of connections of a peer reached by randomly choosing an *edge* of the overlay. This probability is proportional to the degree of the peer ($kp_k$) and it can be proved that its p.g.f. is given by

$$\frac{\sum_k kp_k z^k}{\sum_k kp_k} = z\frac{G'_0(z)}{G'_0(1)} \tag{4.2}$$

where $G'_0(z)$ denotes the first derivative of $G_0(z)$ with respect to $z$ and $G'_0(1)$ yields the average value of distribution $\{p_k\}$. Finally, to characterize the number of connections *excluding* the edge we chose we obtain the p.g.f. from Equation 4.2 by dividing it by $z$:

$$G_1(z) = \frac{G'_0(z)}{G'_0(1)} \tag{4.3}$$

Starting from Equations 4.1 and 4.3 we can compute the p.g.f. for the number of two hops neighbors of a randomly chosen peer as $G_0(G_1(z))$. Similarly, the p.g.f. for three hops neighbor is given by $G_0(G_1(G_1(z)))$, and so on.

For a detailed overview on analyzing generalized random graphs using generating functions, we refer the reader to [Newman 2001].

### 4.3.2 Interconnection of multiple P2P networks

To interconnect multiple overlays we consider some peers as Synapses nodes: these peers belong to multiple P2P networks hence the interconnected system can be modeled by the probability distribution $\{s_i\}$ (with $i \in \{1, \ldots X\}$). The elements of this distribution describe the fraction of nodes belonging to multiple P2P networks: $s_i$ is the fraction of nodes that belong to $k$ P2P networks. Its p.g.f. is given by

$$F(z) = \sum_{i=0}^{\infty} s_i \, z^i \tag{4.4}$$

If we consider one of the $X$ P2P networks including the Synapse nodes then the p.g.f. for the number of connections of a randomly chosen peer can be written as:

$$M(z) = s_1 G_0(z) + s_2 G_0^2(z) + \ldots + s_X G_0^X(z) = F(G_0(z)) \tag{4.5}$$

that is, if the chosen node is a degree 1 synapse (this event has probability $s_1$) then the number of connections is represented by $G_0(z)$. If the node is a degree 2 synapse (this event has probability $s_2$), then the number of connections is represented by the sum of two independent random variables whose p.g.f. is $G_0(z)$; it is well-known that the generating function of the sum of two independent random variables is equal to the product of the respective generating functions yielding the $G_0^2(z)$ factor in Equation 4.5. The same reasoning is valid for synapses whose degree is greater than 2.

A similar expression can be written for the neighborhood of a node reached by following one randomly chosen edge *excluding the selected edge*:

$$\begin{aligned} N(z) &= s_1 G_1(z) + s_2 G_1(z) G_0(z) + \ldots + s_X G_1(z) G_0^{X-1}(z) \\ &= \frac{G_1(z)}{G_0(z)} F(G_0(z)) \end{aligned}$$

If we denote as $N_t(z)$ the p.g.f. for the probability distribution of the number of neighbors $t$ hops away from a randomly chosen node we have that: $N_1(z) = M(z)$, and $N_2(z) = M(N(z))$, and $N_3(z) = M(N(N(z)))$, and so on. From these p.g.f. the average number of neighbors can be computed by evaluating their first derivative w.r.t. $z$ in $z = 1$.

As such, each probability distribution $\{s_i\}$ induces an *interconnection cost* that we define as the average number of P2P networks a randomly chosen node belongs to:

$$f = F'(1) \tag{4.6}$$

### 4.3.3 Search algorithm

To model a flooding-based search in the interconnected system, we consider the set of probabilities $\{p_f(i)\}$, where $i \in \{1, \ldots X\}$. A peer belonging to $i$ overlays sends/forwards a query to a particular neighbor with probability $p_f(i)$, where $i \in \{1, \ldots X\}$). Therefore, $\{p_f(i)\}$ *is not* a probability distribution.

We denote as $q_h$ the probability that $h$ first hop neighbors received a query from the peer that started the search. If the peer belongs to $i$ overlays, it sends a query to one of its neighbors with probability $p_f(i)$. Therefore, the number of neighbors that receive the query follows a binomial distribution with parameter $p_f(i)$. Therefore, it is well known that the probability distribution $\{q_h\}$ has p.g.f. given by [Newman 2001]:

$$\begin{aligned} Q(z) &= s_1 G_0(1+p_f(1)(z-1)) + s_2 G_0^2(1+p_f(2)(z-1)) + \ldots + \\ &\quad + s_X G_0^X(1+p_f(X)(z-1)) \\ &= \textstyle\sum_{i=1}^X s_i G_0^i(1 + p_f(i)(z-1)) \end{aligned}$$

Similarly, for the p.g.f. of the probability distribution describing the number of queries sent by a node reached by following a randomly chosen edge, we obtain:

$$R(z) = \sum_{i=1}^{X} s_i \, G_1(1 + p_f(i)(z-1)) \, G_0^{i-1}(1 + p_f(i)(z-1)) \tag{4.7}$$

If we denote as $Q_t(z)$ the p.g.f. for the probability distribution of the number of neighbors $t$ hops away from a randomly chosen peer that received a query, we have that: $Q_1(z) = Q(z)$, $Q_2(z) = Q(R(z))$, and $Q_3(z) = Q(R(R(z)))$, etc. As a special case, we may consider constant forwarding probabilities, *i.e.* $p_f(i) = p_f, \forall i \in \{1, \dots X\}$. In this case, we would obtain:

$$Q(z) = M(1 + p_f(z-1))$$

and

$$R(z) = N(1 + p_f(z-1))$$

Since the p.g.f. of the probability distribution of the sum of independent random variables is given by the product of the corresponding p.g.f., the total number of queries generated by a search issued by a randomly chosen peer is described by:

$$T(z) = \prod_{t=1}^{TTL} Q_t(z)$$

yielding the *average number of queries*

$$m = T'(1). \tag{4.8}$$

### 4.3.4 Hit probability

We model resource popularity by $0 \leq \alpha \leq 1$ that is the average fraction of peers that globally hold the given resource. We interpret this parameter as the probability that a randomly chosen node owns a copy of the resource.

If we denote as $w_h$ the probability that $h$ first hop neighbors hold a copy of the requested resource *and* received a query from a peer that belongs to $i$ overlays we note that the number of such neighbors follows a binomial distribution with parameter $\alpha p_f(i)$. If we denote as $H_t(z)$ the p.g.f. for the probability distribution of the number of neighbors $t$ hops away from a randomly chosen peer that received a query *and* hold a copy of the requested resource then we have that: $H_1(z) = Q_1(1 + \alpha(z-1))$, $H_2(z) = Q_2(1 + \alpha(z-1))$, $H_3(z) = Q_3(1 + \alpha(z-1))$, and so on. Therefore, the total number of search hits is described by a probability distribution whose p.g.f. is given by:

$$H(z) = \prod_{t=1}^{TTL} H_t(z)$$

yielding the search *hit probability*

$$p_{hit} = 1 - H(0) \tag{4.9}$$

### 4.3.5 A variation of the search algorithm

To model a search algorithm where peers that own a copy of the resource do not forward a query message it suffices to redefine $R(z)$ in Equation 4.7. In particular, when a peer owns a copy of the resource the number of its neighbors that receive the query is equal to 0: this happens with probability $\alpha$. In Equation 4.10 this is represented by the term $\alpha$ that can be written as $\alpha p_0 z^0$ with $p_0 = 1$. With probability $1-\alpha$ Equation 4.7 holds, therefore we obtain the p.g.f. of the probability distribution describing the number of queries sent by a node reached by following a randomly chosen edge as:

$$\begin{aligned} R(z) = {} & \alpha + \\ & + (1-\alpha)\sum_{i=1}^{X} s_i G_1(1+p_f(i)(z-1))G_0^{i-1}(1+p_f(i)(z-1)) \end{aligned}$$

The definition of $Q_t(z)$, and $T(z)$, and $m$ remains unchanged.

## 4.4 Results

In this section, we will first show the results of the model validation, performed via a heavily multi-threaded simulator, written in Erlang [USy ], that reproduces, in terms of message routing, the exact behavior of a system described by our model. Also, we will show the results of some broad system evaluations made possible by the use of our model to compute metrics that would otherwise, if performed by means of simulations, require too much in terms of simulation time and computational power.

In our analysis, we consider different routing policies that can be employed in our scenarios, modeled by defining the $p_f(i)$ mentioned in Section 7.2. Those are:

- $p_f(i) = \frac{1}{i}$, henceforth referred to as *1/i*, i.e. the probability of selecting a neighbor is inversely proportional to the number of overlays a node is connected to. This routing tends to maintain a constant number of messages, but "flattens" the interconnected topology, not allowing synapse nodes to exploit the extended neighborhood.

- $p_f(i) = min(1, \frac{z_{max}}{zi})$, henceforth referred to as $zmax$, where $z = E[\{p_k\}]$ is the average number of neighbors for a node based on the current degree distribution and $z_{max}$ is a system parameter, specified upon design, indicating the upper bound for the average number of forwarded messages. This policy allows for a better exploitation of Synapse nodes, while still finely limiting the number of messages in the system. In our evaluations, $z_{max}$ has been set to $2z$, twice the average number of neighbors per node.

- $p_f(i) = 1$, henceforth referred to as *flood*, i.e. a routing where every node selects forwards a message to *every* neighbor, regardless of the number of connected overlays.

In both simulations and evaluations, the individual overlays have been modeled following the neighbors degree distribution measured in [Bolla 2009] from real world applications and used already in [Gaeta 2011], in order to have an accurate overlay model.

### 4.4.1 Model validation

In order to evaluate the accuracy of our model in predicting the performance indexes of a real network, we validated the obtained results by means of simulation. The simulator (available at [USy ]) employs standard statistical procedures to estimate 68% and 95% confidence intervals for the $p_{hit}$ and $m$ indexes defined in Section 4.3.

#### 4.4.1.1 Simulation methodology

The simulator has been developed from scratch in Erlang [erl ]. The choice of Erlang has been driven by its native multi-threading capabilities and inter-process communication model based on the message passing paradigm embedded in the language, thus allowing for a rapid implementation of an accurate network model made of node processes running independently and exchanging messages with one another. Each process has a list of other processes it can exchange messages with, that constitutes its neighborhood.

We consider $N_s$ independent realizations for the interconnected overlay topologies (in our experiments $N_s = 30$); each interconnected topology is used to obtain one realization of $m$ and $p_{hit}$. The $h^{th}$ realization is obtained as follows:

- We first generate a new topology, made of $X$ overlays interconnected by synapse nodes, using as input parameter the number of nodes $N = 500000$, the nodes degree distribution $\{p_k\}$ [Bolla 2009], and the $\{s_i\}$ to be validated;
- From the generated topology file, the simulator instantiates $N$ node processes and assigns each the corresponding list of neighbors;
- One or more resources are then seeded in the system, according to their respective popularity $\alpha$, by sending a `PUT(value)` message to $N\alpha$ random nodes;
- Separate worker processes take care of sending a query message `SEARCH(value,TTL)` to each node process in the network.
- Meanwhile, a listener process receives then the responses, either the resource being found or the $TTL$ being reached, and of computes the statistics.

#### 4.4.1.2 Topology generation

The generation of a network made of interconnected overlays mainly consists of generating first $X$ individual overlay topologies, and then connecting them by “merging” nodes from different overlays in one Synapse node, thus creating nodes with extended neighborhoods spanning across all the connected overlays. In order to generate $X$

random graphs with a specified degree $\{p_k\}$ we relied on the algorithm presented in [Viger 2005], that provides short generation times while guaranteeing the respect of the specified degree.

#### 4.4.1.3 Validation results

The first validation we performed was conducted for a system with only one overlay ($X = 1$). For the sake of brevity we only show the results for the *flood* routing strategy, $\alpha = 0.0001$, and $TTL = 3$. Table 4.2 shows the model is very accurate and faithfully predicts results when compared to the simulation output.

We then validated various scenarios with a higher number of interconnected overlays ($X = 4$), at $TTL = 3, 4$ and with different values of $\alpha$, different routing policies and different distributions $\{s_i\}$. We considered the distribution for the degree of synapses summarized in Table 4.3.

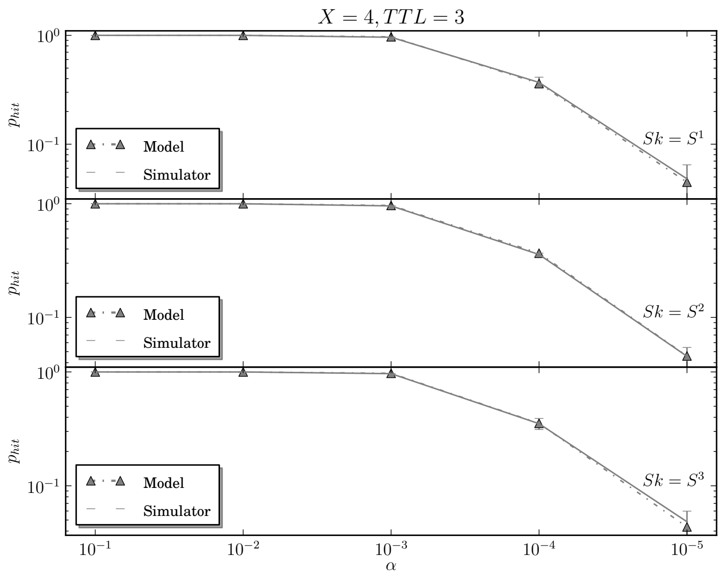


Figure 4.2: $p_{hit}$ (with confidence interval) for different $\alpha$ and $s_i$ distributions: comparison between model and simulation.

Figure 4.2 shows a comparison between the computed $p_{hit}$ for different values of

Table 4.2: $m$ for different $s_i$ distributions: comparison between model and simulation.

| | **Model** | **Simulation (95% C.I.)** |
|---|---|---|
| $p_{hit}$ | 0.3733 | $0.373552 \pm 0.003852$ |
| $m$ | 4822.63 | $4821.57 \pm 0.0498$ |

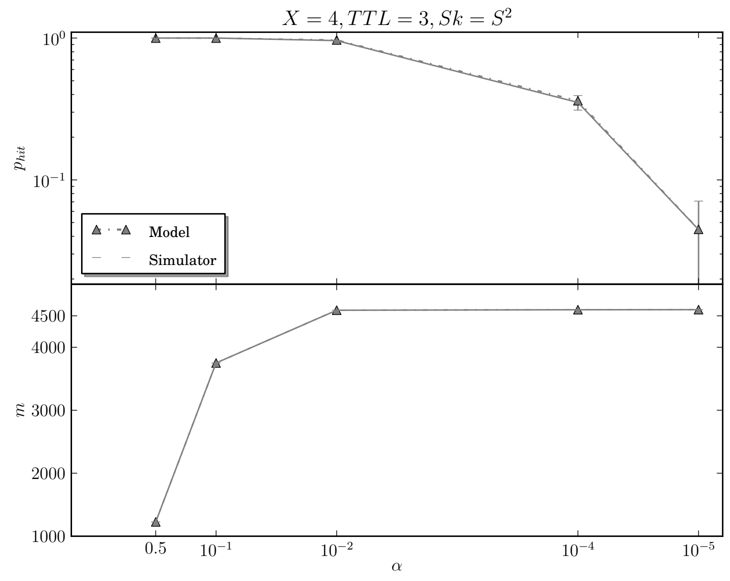


Figure 4.3: Alternative search algorithm (Section 4.3.5): comparison between model and simulation.

$\alpha$ and the corresponding simulation results, while Table 4.4 summarizes the same comparison for $m$. The results show how both performance metrics computed by our model fall within the confidence interval of the simulation results.

Furthermore, we validate the system against the alternative search algorithm detailed in Section 4.3.5. For the sake of brevity, we are showing results only for $S^2$ since the same conclusions can be drawn for $S^1$ and $S^3$. Figure 4.3 shows both $p_{hit}$ and $m$ against different values of $\alpha$, since with this algorithm the number of message is dependent of the resource popularity. Even in this scenario, the model results fall within the confidence interval estimated by the simulator.

Therefore, we can safely conclude that our model is accurate in predicting the behavior of the performance indexes we defined in a broad range of different scenarios. Furthermore, while simulations required hours of CPU time to complete solving our model took less than a second with a solver implemented in C.

Table 4.3: Definition of the $\{s_i\}$ distributions used for validation.

| | |
|---|---|
| $S^1$ | $s_1 = 0.7, s_2 = 0.1, s_3 = 0.1, s_4 = 0.1$ |
| $S^2$ | $s_1 = 0.4, s_2 = 0.3, s_3 = 0.2, s_4 = 0.1$ |
| $S^3$ | $s_1 = 0.1, s_2 = 0.2, s_3 = 0.3, s_4 = 0.4$ |

### 4.4.2 Model exploitation

After validating the model we conducted a few analysis to show its usefulness in the design phase of the interconnection of several peer-to-peer networks.

#### 4.4.2.1 Comparison of different routing policies

A first evaluation concerns the choice of a specific routing policy in the system, i.e. the definition of different $p_f(i)$. In this case, we want to compare for values of $\alpha$ down to $10^{-6}$, the performances in terms of $p_{hit}$ and $m$ for the distribution of degree of synapses $S^1$ (results for the other two distributions suggested similar considerations and are omitted for the sake of brevity), $X = 10$, and $TTL = 3$. Please note that to achieve a reliable measurement via simulation for $\alpha = 10^{-6}$ we would need to conduct complex simulations (at least 1000000 nodes) for a long simulation time (ideally each of them to be queried individually for multiple topology realizations).

Figure 4.4 show the values of $p_{hit}$ for the 3 different policies and different resource popularities, while Figure 4.5 depicts the average number of messages for the 3 policies in the case of propagation of queries up to $TTL$ hops (Figure 4.5(b)) and for the query propagation that stops when reaching a node holding a copy of the resource (Figure 4.5(a)) modeled in Section 4.3.5. In the former case, the number of messages is independent of the resource popularity while in the latter case we note that reduction of the number of query messages can be obtained for popular resources, i.e., for $\alpha > 0.01$.

In this case, the model allows for a simple cost/benefit evaluation, based on the expected popularity of a resource. For one, we can notice an almost tenfold increase in the number of messages between the *zmax* and the *flood* policy, to which it does not correspond a proportional increase in the $p_{hit}$.

#### 4.4.2.2 $f$-cost based evaluation

In a cost/benefit analysis of the interconnected system, we consider $p_{hit}$ as our benefit metric whereas $m$ and $f$ are considered as costs. Another kind of evaluation we performed consists of fixing the $f$ cost and analyzing which distributions $\{s_i\}$ lead to better performances ($p_{hit}$) and minimum cost ($m$).

To this end we considered all distributions $\{s_i\}$ that can be defined for $X = 5$ where the individual probabilities are non-zero multiple of 0.05. We considered 3

Table 4.4: $m$ for different $s_i$ distributions: comparison between model and simulation.

| | **Model** | **Simulation (95% C.I.)** |
|---|---|---|
| $S^1$ | 4598.02 | $4596.77 \pm 2.38$ |
| $S^2$ | 4701.82 | $4700.96 \pm 0.49$ |
| $S^3$ | 4449.57 | $4453.58 \pm 3.41$ |

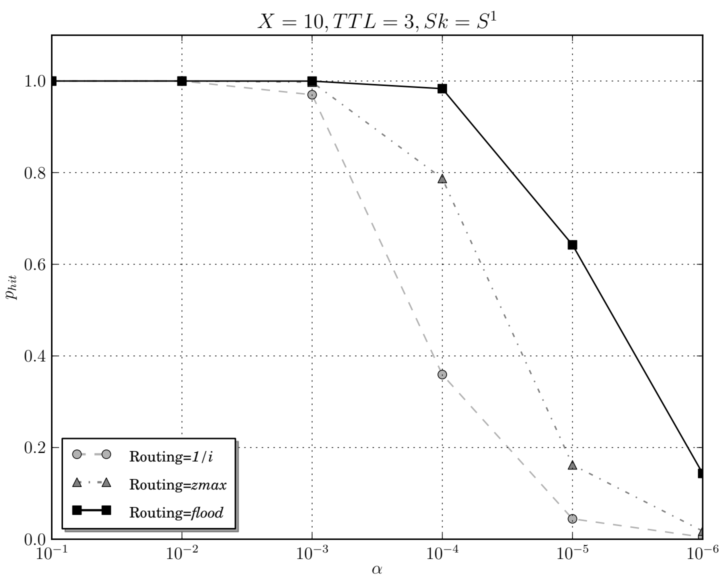
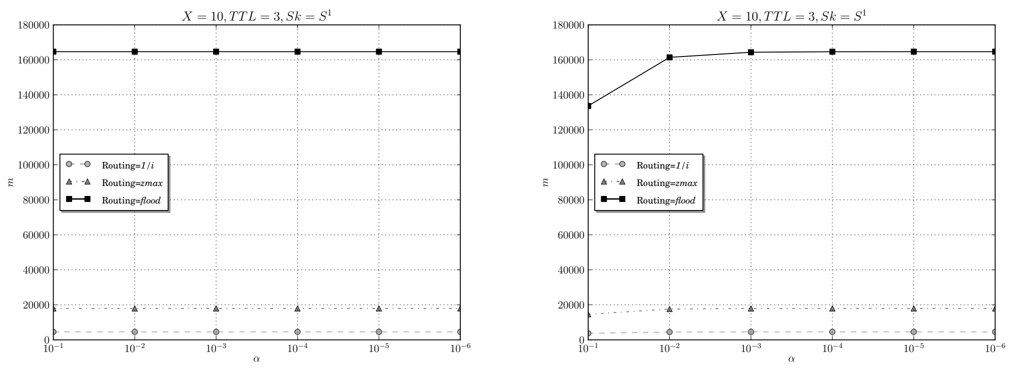


Figure 4.4: Routing policies comparison: $p_{hit}$ for different resource popularities $\alpha$.

values of $f$ (namely, $f = 2, 3, 4$) and compared the performances of every distribution $\{s_i\}$ with given $f$ for $TTL = 2$. Again, please note that this analysis would have required days of CPU time to be completed by means of simulation since even with a coarse granularity in the definition of $\{s_i\}$ (0.05) we tested hundreds of different distributions. This analysis required only a few seconds to complete with our model.

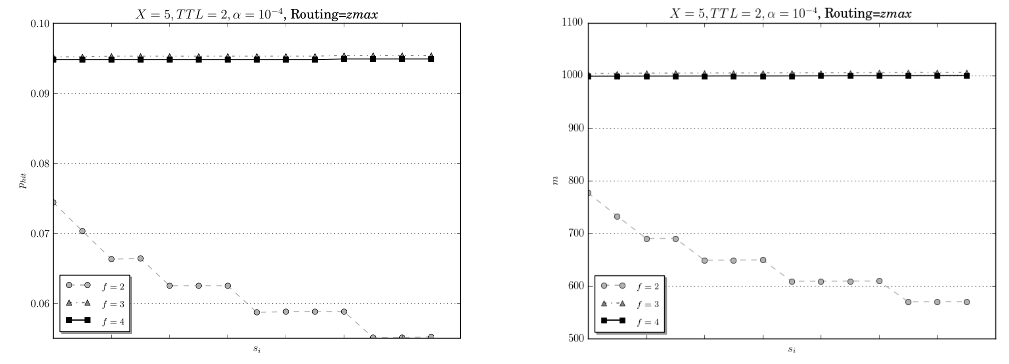
Figures 4.6(a) and 4.6(b) show a subset of these distributions (each point in the graph corresponds to a particular distribution $\{s_i\}$). We only plotted the ones with the highest $p_{hit}$; it appears that the interconnection cost $f$ alone is not directly bound to an increase in performances. There are, as a matter of fact, different configurations with $f = 3$ that perform equally (sometimes very slightly better) than those with a $f = 4$. Furthermore, within the configuration with $f = 2$ some are better than others in terms of performance and costs. Nevertheless, a clear relation exists between message cost $m$ and $p_{hit}$: the larger the average number of messages the higher the $p_{hit}$.

The behavior shown in the figures can be explained as following: the routing policy *zmax* limits the number of messages that can be issued by a node to $z_{max}$, which is set in our evaluations to $2z$. Therefore, increasing the number connections in the interconnected system ($f$) beyond certain values does not lead to a significant performance increase. That is why we observe a proportionally higher increase in the $p_{hit}$ from $f = 2$ to $f = 3$ than from $f = 3$ to $f = 4$.



(a) Query propagation for $TTL$ hops (b) Query propagation of Section 4.3.5

Figure 4.5: Average number of messages for different routing policies.



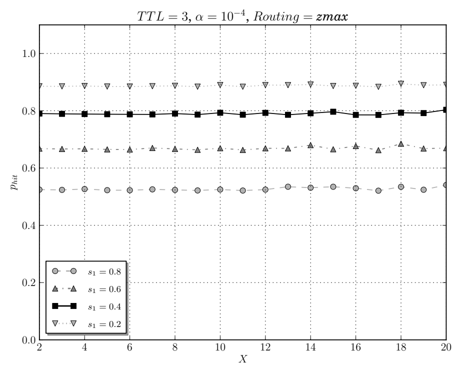
(a) Hit probability $p_{hit}$ (b) Number of messages $m$

Figure 4.6: $s_i$ comparison at different $f$.

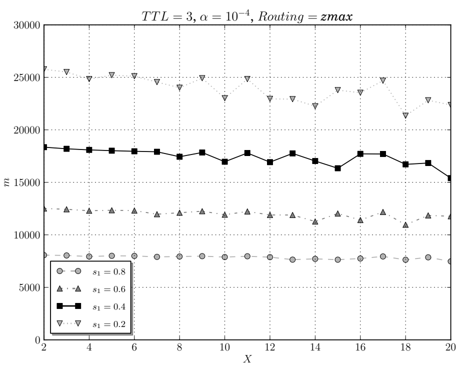
#### 4.4.2.3 Effects of granularity

Another aspect we analyze is a performance comparison as the number of overlays to interconnect increases. In this case we chose to analyze the behavior of the *zmax* routing policy, in a system with $TTL = 3$ and $\alpha = 0.0001$, for an increasing number of overlays ($X$) and for different distributions $\{s_i\}$, characterized by an increasing percentage of non-synapse nodes $s_1$, while the remainder of the distribution is equally distributed across the remaining $s_i$.

Figures 4.7(a) and 4.7(b) show four different configurations, with an increasing number of non-synapse nodes in the system. The parameter $s_1$ indicates the share of non synapses nodes, while the remaining part $(1 - s_1)$ is equally distributed among the remaining $X - 1$ values, i.e., $s_i = \frac{1-s_1}{X-1}$ for $1 < i \leq X$. It can be noted that at each given ratio of synapses vs non-synapses nodes the system behavior is roughly the same regardless the number of overlays. The efficiency is still tightly bound to

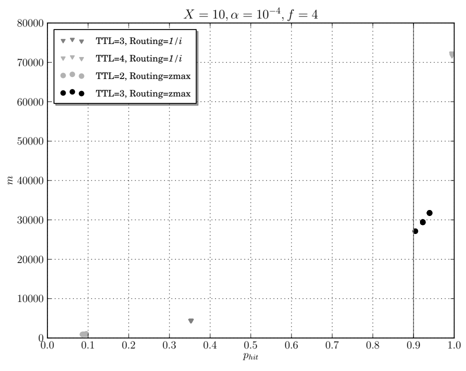


(a) $p_{hit}$ vs. $X$

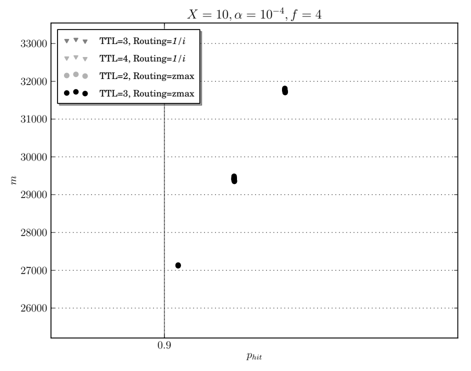


(b) $m$ vs. $X$

Figure 4.7: Performance evaluation with different numbers of overlay $X$.



(a) Overall view



(b) Zoomed view

Figure 4.8: Distribution of different routing policies with fixed $f$.

the number of messages and both increase as $s_1$ decreases.

#### 4.4.2.4 System design with minimum requirements

Thanks to the high number of different configurations that can be evaluated with our model in a relatively short time, we conduct a further analysis to support the design of the interconnection of several peer-to-peer networks.

For instance, we set the number of overlays $X$ and the resource popularity $\alpha$; by setting a bound for the minimum desired $p_{hit}$, we can compare different routing policies and $TTL$ values and find the one that minimizes the average number of messages $m$.

Figures 4.8 and 4.9 show a classification of distributions $\{s_i\}$ for two different routing policies and two different $TTL$ values with respect to $p_{hit}$ and $m$ for $X = 10$ and $\alpha = 0.0001$ (each point in the graphs represents a particular distribution

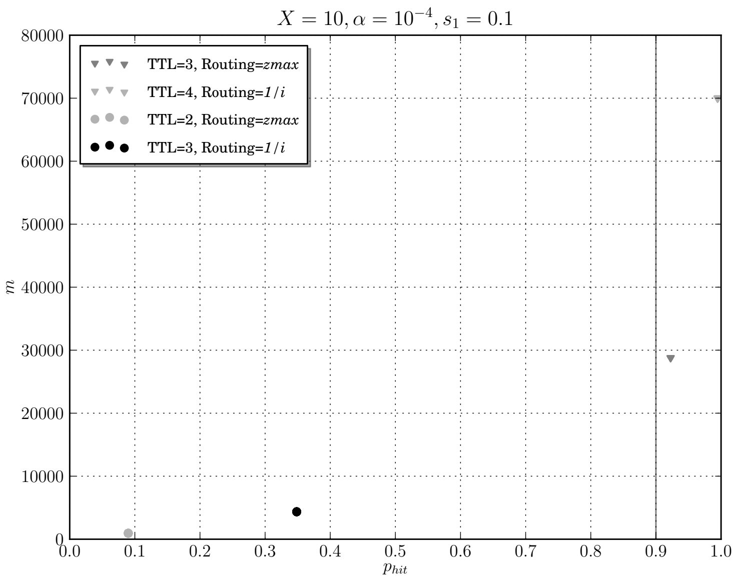


Figure 4.9: Distribution of different routing policies with fixed $s_1$.

$\{s_i\}$). In the first case (Figure 4.8), we decided to fix a cost factor and set $f = 4$, whereas in the second case (Figure 4.9), the fixed factor is the ratio of expected non-synapse nodes in the system $s_1$. We are able to discriminate immediately those distributions $\{s_i\}$ that do not satisfy the imposed criteria of having $p_{hit} > 0.9$. We also discriminate among those that do the distributions $\{s_i\}$ that minimize the number of messages $m$, as shown in Figure 4.8(b).

#### 4.4.2.5 Routing without propagation

We briefly present some evaluation results based on the model variation presented in Section 4.3.5. In the first version of our model, the routing of a message is assumed to continue until the TTL expires, regardless of a resource being found or not. This leads to an $H_t(z)$ able to describe different cases, such as the probability of finding *multiple* copies of a resource. However the system is not optimal message-wise. In case we are interested only in the first hit of a search query, and we want to optimize the number of messages employed, with the variant of $R(z)$ described in 4.3.5 we are able to evaluate the system under the conditions that the routing in a node stops whenever a resource is found.

Figure 4.10 shows the trend of $m$ for different $\alpha$, and two routing policies for $X = 10, TTL = 3$, and distribution $S^1$. While the number of messages was unrelated to the resource popularity before, here we see that, as routing stops upon first hit, the more popular a resource, the lower the number of messages per query.

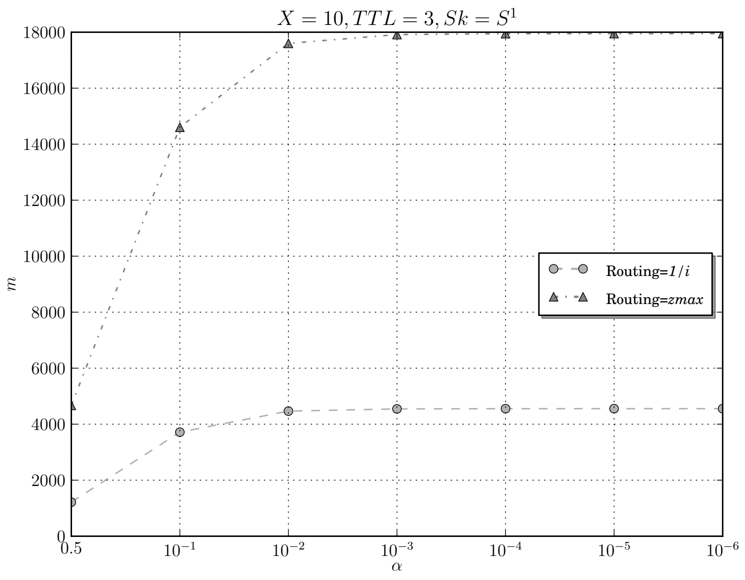


Figure 4.10: Message evaluation at different $\alpha$, for different routing policies.

## 4.5 Conclusion

In this chapter we considered a mathematical modeling for large scale unstructured P2P networks interconnected to one another via co-located nodes called Synapses: these nodes send/forward a query to all the P2P networks they belong to. We developed a generalized random graph based model to represent the topology of one unstructured P2P network, the partition of nodes into synapses, the probabilistic flooding based search algorithms, and the resource popularity. We validated our model against simulations and proved that its predictions are reliable and accurate. The model allowed the analysis of very large and complex systems: we believe that simulation and/or prototype deployment based analysis would be unfeasible in this case.

The contributions of this chapter have been published as
*Modeling and Analysis of Large Scale Interconnected Unstructured P2P Networks*, Poster paper, in Proceedings of the 18th International Conference on Parallel and Distributed Computing (ICPADS), IEEE [Ciancaglini 2012a] and
*Interconnection of large scale unstructured P2P networks: modeling and analysis* in Proceedings of the Twentieth International Conference on Analytical and Stochastic Modelling Techniques and Applications, Springer LNCS [Gaeta 2013].

# Part II

# Applications on top of interconnected overlays

Chapter 5

# CarPal: an example of social crowdsourced application

## Contents

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 ability of the community to effectively share and retrieve information about travelers and itineraries. Structured overlay networks, such as Chord [Stoica 2001] or Kademlia [Maymounkov 2002a], have emerged recently as a flexible solution to handle large amounts of data without the use of high-end servers, in a decentralized manner. In this chapter, 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 for deployment on portable devices, and opens new scenarios in which trips and sharing requests can be updated in real time. Furthermore, the interconnected architecture described in Chapters 2 and 3 is leveraged to allow for the interconnection of nearby communities, with a higher probability of common travel patterns between their members, thus allowing for the increase of a query's success rate, the number of effectively shared rides and the effectiveness of our solution.

## 5.1 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, e.g. home-to-work or home-to-school). Amongst its 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 global dependency on petrol.

In Car pooling services, an Information System (IS) has been shown 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 to 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 their offers/requests. In presence of multiple services, for technical and/or commercial reasons, it is not possible to share content across different providers, despite the evident advantage. As a simple example, the reader can have a quick look on Equipage06 [a] and OttoEtCo [d], two websites concerning car pooling in the French Riviera. At the moment the two do not communicate, share any user profile nor requests, even if they operate on the same territory and with the same objectives. Since both services are non-profit, the reason for this lack of cooperation would probably 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 would make both services unavailable if both servers were to be down. With this in mind, in this chapter we propose and implement a peer-to-peer based Carpool information system, which we call *CarPal*: this service is suitable for deployment in a very low infrastructure and can run on various devices, spanning from PCs to small intelligent devices, like smart phones. The system exploits the Synapse Framework, presented in Chapters 2 and 3, in order to allow two completely independent CarPal-based communities to exchange information with one another, without the need of merging one community into the other or, even worse, build a third CarPal system including both.

## 5.2 Application architecture

### 5.2.1 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 quickly 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 their own travel database (which, as it will be shown later, can be interconnected with sibling communities) and manage it in a distributed manner. Furthermore, it constitutes a flexible infrastructure within which, by deployment on connected mobile devices, it will be possible to develop more advanced info-mobility solutions which might take into account the position of the user/vehicle (via an internal GPS), geographically-aware network discovery or easy network join, or vehicle tracking through checkpoints with the use of Near Field Communications technologies [c].

### 5.2.2 CarPal in a nutshell

A user running CarPal on his mobile device or desktop computer can connect to one or more communities of which he is member (i.e. he has been invited or a request of his has been accepted). Two operations would then be 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 of finding 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 of a recurring trip;
- *Place of departure and place of arrival*, whose representation is critical, since high granularity might lead to the omission of similar results;
- *Time of departure*;
- *Time of arrival* 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. pet allergies, other specific needs etc.;

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

Once the user has inserted all the required data (date, place and time of departure and arrival, number of seats and optional checkpoints), the trip is decomposed to all possible combinations: for example, a trip containing the stops 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; furthermore all of the segments which do not start from 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, while the others are subject to traffic conditions and contingencies. Geographic and time information must be encoded in such a way that it is precise enough to still be relevant for our purposes (someone leaving from the same city but 10 km far is not a useful match) yet broad in the sense that a precise query will not omit any relevant results.

Every checkpoint (including departure and arrival point) could either be inserted directly through geographical coordinates (using the GPS capabilities of modern mobile devices) or as an address that would then be converted in geographical coordinates using Reverse Geo-location APIs made available by services such as Google Maps [b]. Such coordinates would then be rounded before the hash key encoding in order to group together locations within a given radius (around 5 kilometers). Concerning time approximation, a 20-minute-window is used to approximate departure times. Both during an insertion or a query, anything within the 0-19 minute interval would 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 trips 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). Furthermore, 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 would 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 would be notified through the application. The individual trip records will then be updated by decreasing the number of available seats.

| Key | Value | Grouping criteria |
|---|---|---|
| "I" ⌣TRIP_ID | ♣ | Individual trip |
| "T" ⌣DATE ⌣DEP ⌣TOD ⌣ARR ⌣TOA | list[TRIP_ID] | Departure, Arrival & Time |
| "B" ⌣DATE ⌣DEP ⌣ARR | list[TRIP_ID] | Departure & Arrival |
| "D" ⌣DATE ⌣DEP | list[TRIP_ID] | Departure |
| "A" ⌣DATE ⌣ARR | list[TRIP_ID] | Arrival |
| "U" ⌣USER_ID | list[TRIP_ID] | User |

where ♣ = [DATE,DEPARTURE,TOD,ARRIVAL,TOA,SEATS,CONTACT,PUBLIC]

Table 5.1: Keys updated in the DHT for each new entry

### 5.2.3 Encoding CarPal in a DHT

The segments are stored in the DHT according to Table 5.1. The "⌣" symbol represents, with a little abuse of notation, the concatenation of multiple values for one key. Multiple keys, representing different sets of trips grouped according to different criteria, are updated for each entry (or created if they do no already exist), namely:

1. The actual trip record, associated to a unique TRIP_ID, that will be updated, e.g., when someone books a seat. The information stored concerns trip date - DATE, place and time of departure - DEPARTURE and TOD, place and time of arrival - ARRIVAL and TOA, number of available seats (or cargo space, in case of shared goods transportation) - SEATS, a reference to contact the driver - CONTACT, and if the trip has to be public or not - PUBLIC. Depending on the needs more information can be appended to this record; the key is created by appending the token "I" to the TRIP_ID

2. The set of trips having the same date, place and time of departure and arrival. The key is created by concatenating the token "T", trip date - DATE, place and time of departure - DEPARTURE and TOD, place and time of arrival - ARRIVAL and TOA. Its value is a list of TRIP_ID pointing to the corresponding trip records.

3. The set of trips grouped by date and place of departure and arrival. 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 to the token "B" the trip date, place of departure and place of arrival;

4. The sets of trips arranged by day and by point of departure or arrival. The key is therefore made by concatenating either the token "D" (for departure) or "A" (for arrival) to the date - DATE and point of departure or arrival - DEP or ARR. This set 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. The set of trips for a given user. The key is the token "U" prepending the USER_ID itself.

### 5.2.4 Network architecture

The overlay chosen for the proof of concept is Chord [Stoica 2001] although other protocols could be used to exploit the locality of the application or a more direct geographical mapping. 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 summed up as follow:

- It allows for the setup of a new community, by registering the IP of certain reliable peers, in a YOID-like fashion [Francis 2000];
- 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 an entry point for downloading the application and getting updates.

## 5.3 A Running example

We hereby present a first proof-of-concept for a CarPal application implementing the concepts discussed above. A basic user interface is proposed, showing a first attempt to integrate a mapping service (namely, Google Maps [b]) in the application to render the user experience more pleasant and efficient, although no GPS capabilities and no reverse geolocation are in place yet.

### 5.3.1 Building the scenario

Let us turn to a practical example in order 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 5.1) 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 (such as Antibes, Nice and Cagnes sur Mer).

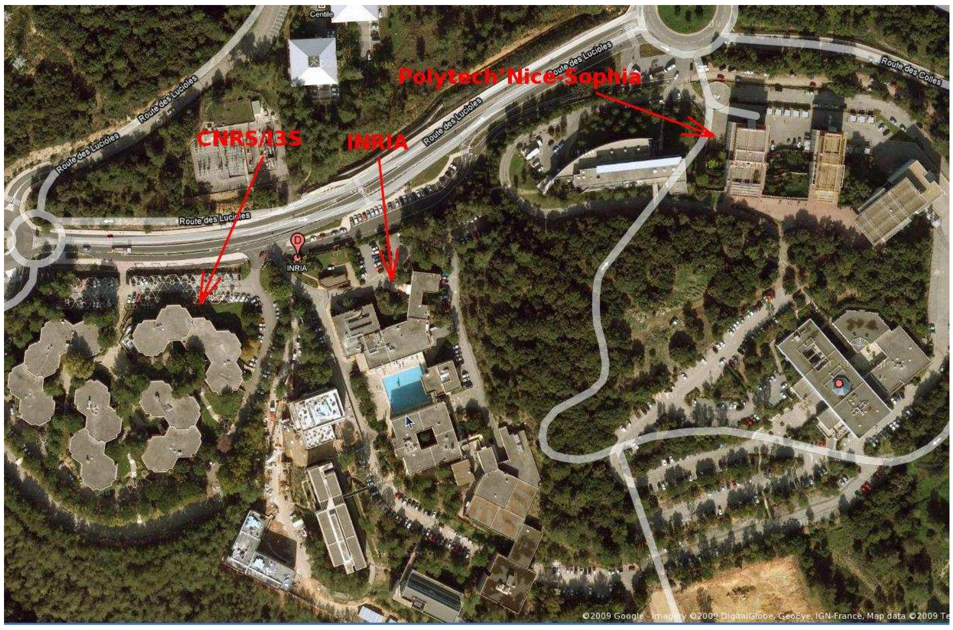

Polytech'Nice-Sophia
CNRS/I3S
INRIA
Route des Lucioles
Route des Colles
INRIA
©2009 Google - Imagery ©2009 DigitalGlobe, GeoEye, IGN-France, Map data ©2009 Te

Figure 5.1: The geographical set-up

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 5.4).

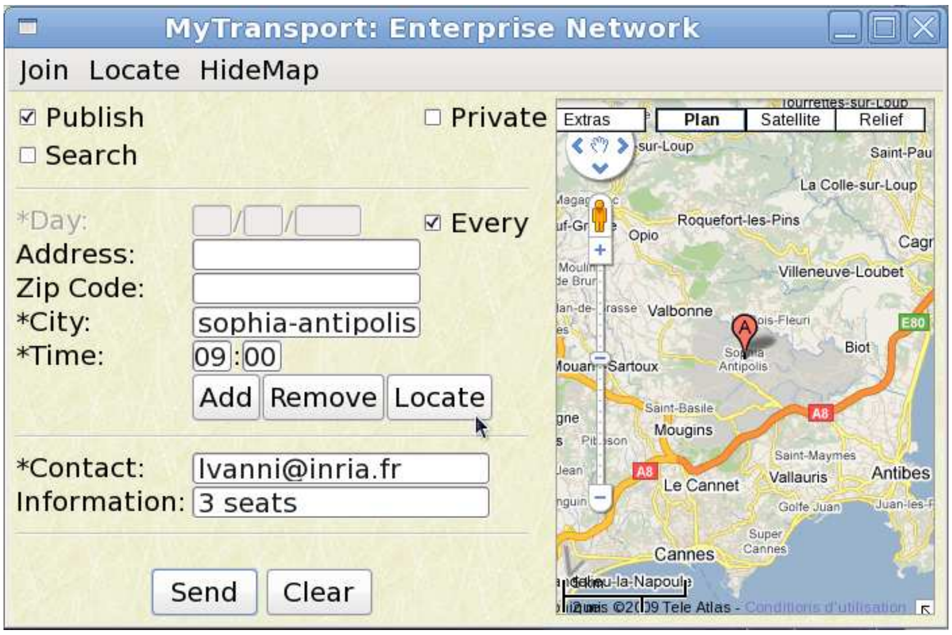
As previously described, there is a checkpoint where our user is willing to stop and pick up some passengers.

| | |
|---|---|
| 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 |

Figure 5.2: Journey data

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

Figure 5.3: Sliced & diced segments



| | |
|---|---|
| ☑ Publish | ☐ Private |
| ☐ Search | |
| *Day: | / / ☑ Every |
| Address: | |
| Zip Code: | |
| *City: | sophia-antipolis |
| *Time: | 09:00 |
| *Contact: | lvanni@inria.fr |
| Information: | 3 seats |

Figure 5.4: CarPal application publishing a new trip

### 5.3.2 Slice and Dice and encoding in the DHT

Starting from the above data all of the possible combinations are generated leading to the segments shown in Figure 5.2 and 5.3. Only the differences are reported, with each of those segments sharing the same date, number of available seats and contact information. The 3 segments are then stored in the DHT by updating (or adding) the appropriate keys as shown in Table 5.2. For clarity purposes, in Table 5.2, date and time values are represented as strings and instead of the actual geographic coordinates a placeholder is shown (i.e. NICE, SOPH...).

A PUT operation represents the insertion of a not yet existing key whereas the APPEND operation assumes that the key might already be 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 stands available to be searched. From Figure 5.4 we can see that it is possible to set the option of the the trip staying private. In that case, the segments will be discoverable only by members of the same network.

| **Operation** | **Key** | **Value** |
|---|---|---|
| PUT | “I”␣123 | ♣ |
| PUT | “I”␣124 | ♠ |
| PUT | “I”␣125 | ■ |
| APPEND | “T”␣20100115␣NICE␣0800␣SOPH␣0900 | 123 |
| APPEND | “T”␣20100115␣NICE␣0800␣CAGN␣0830 | 124 |
| APPEND | “T”␣20100115␣CAGN␣0830␣SOPH␣0900 | 125 |
| APPEND | “B”␣20100115␣NICE␣SOPH | 123 |
| APPEND | “B”␣20100115␣NICE␣CAGN | 124 |
| APPEND | “B”␣20100115␣CAGN␣SOPH | 125 |
| APPEND | “D”␣20100115␣NICE | 123 |
| APPEND | “D”␣20100115␣NICE | 124 |
| APPEND | “D”␣20100115␣CAGN | 125 |
| APPEND | “A”␣20100115␣SOPH | 123 |
| APPEND | “A”␣20100115␣CAGN | 124 |
| APPEND | “A”␣20100115␣SOPH | 125 |
| 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]

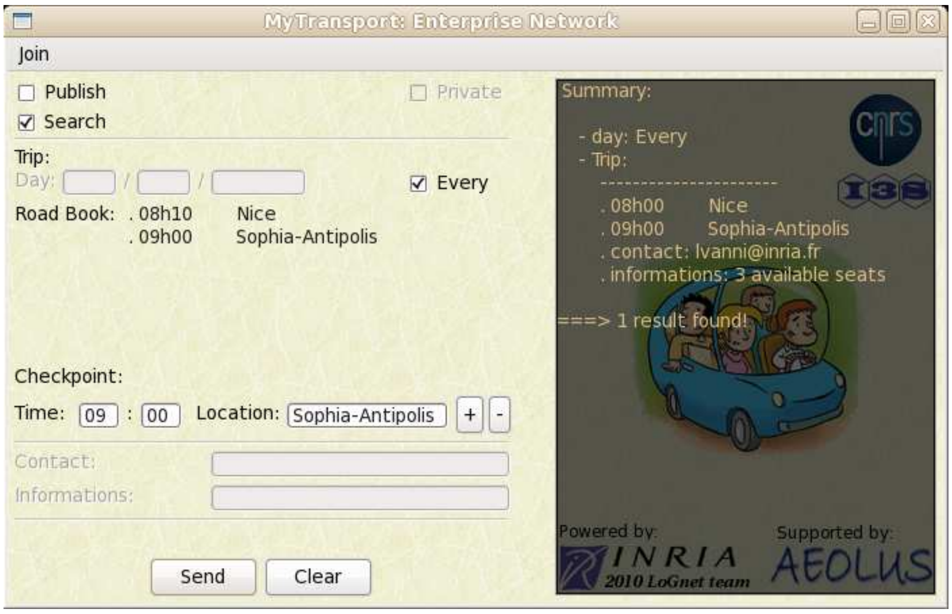
Table 5.2: DHT operations

### 5.3.3 Searching for a trip

A search for a trip follows a similar path as the trip submission. As we can see in Figure 5.5 the user can specify an itinerary, a specific time and even some intermediate segments, in order 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 Point of Departure and Arrival to browse through the day's trips or a key with only Departure or Arrival for a broader search. Thanks to the Slice and Dice operation, it is possible to aggregate segments coming from different users as Figure 5.6 shows.

In 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 (which has the same functions and technology).

By marking his published itinerary as public, a member of the Enterprise Network allows the students to get matching results via a synapse (Figure 5.7), i.e. somebody registered to both networks (Figure 5.8). This allows the system to increase the chances of finding an appropriate match while maintaining good locality properties (Figure 5.9).



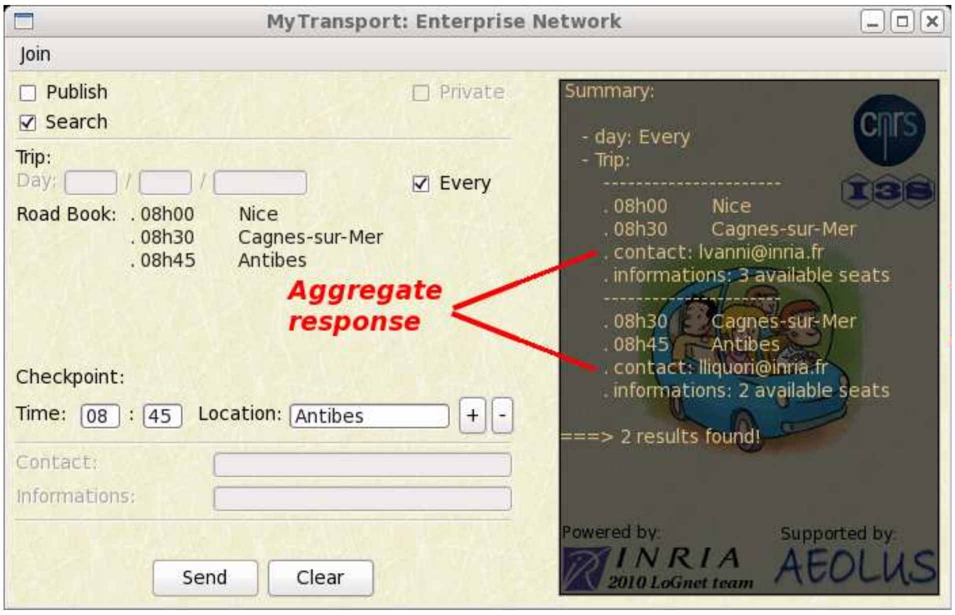
Trip:
Day: / /
Road Book: . 08h10 Nice
. 09h00 Sophia-Antipolis

Checkpoint:
Time: 09 : 00 Location: Sophia-Antipolis
Contact:
Informations:

Summary:
- day: Every
- Trip:
---------------------
. 08h00 Nice
. 09h00 Sophia-Antipolis
. contact: lvanni@inria.fr
. informations: 3 available seats

===> 1 result found!

Figure 5.5: Simple search



Trip:
Day: / /
Road Book: . 08h00 Nice
. 08h30 Cagnes-sur-Mer
. 08h45 Antibes

Checkpoint:
Time: 08 : 45 Location: Antibes
Contact:
Informations:

Summary:
- day: Every
- Trip:
---------------------
. 08h00 Nice
. 08h30 Cagnes-sur-Mer
. contact: lvanni@inria.fr
. informations: 3 available seats
---------------------
. 08h30 Cagnes-sur-Mer
. 08h45 Antibes
. contact: lliquori@inria.fr
. informations: 2 available seats

===> 2 results found!

Figure 5.6: Aggregate results

## 5.4 Conclusion

In this chapter we presented a first example of how the interconnected architecture presented earlier in this thesis can be exploited to develop an application fulfilling a

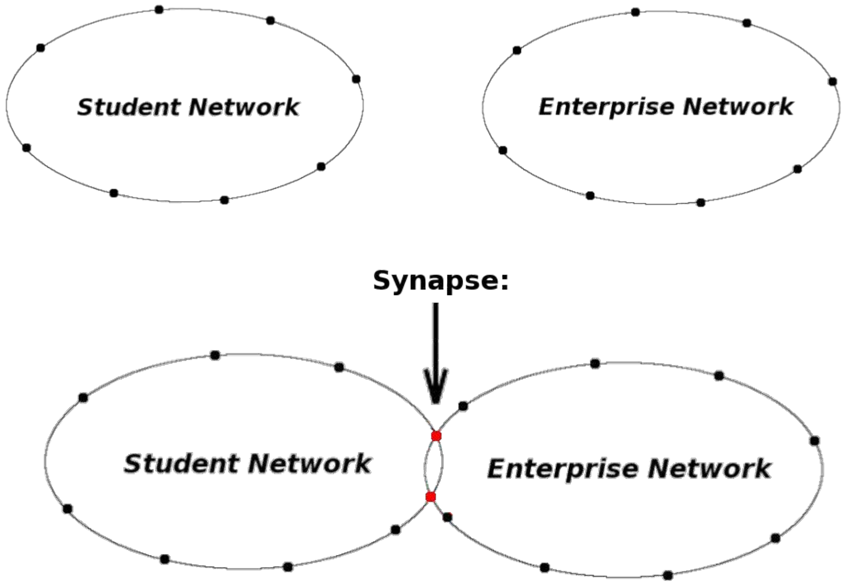


Figure 5.7: Students, Enterprise and Synapsed Overlay Networks

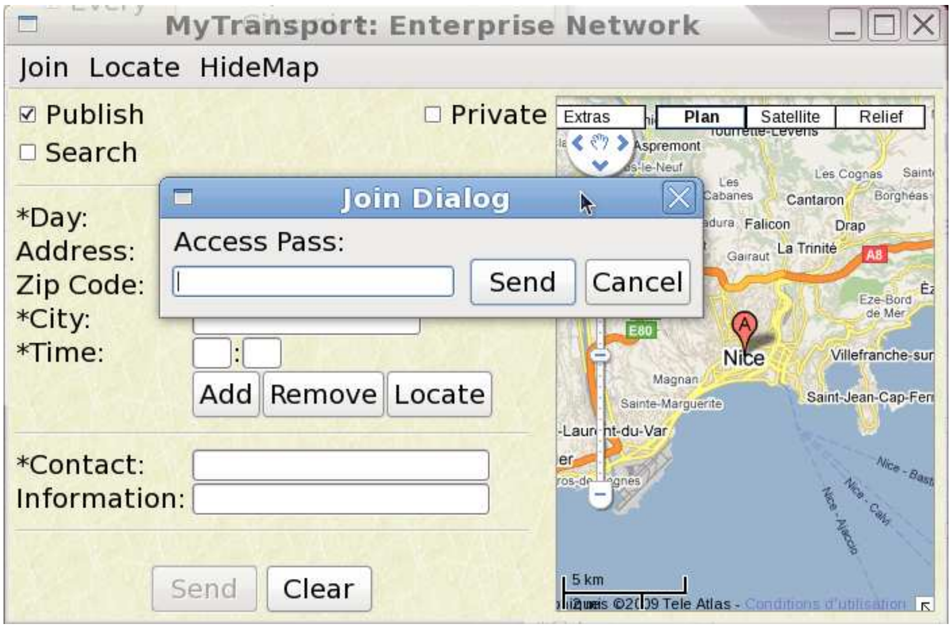
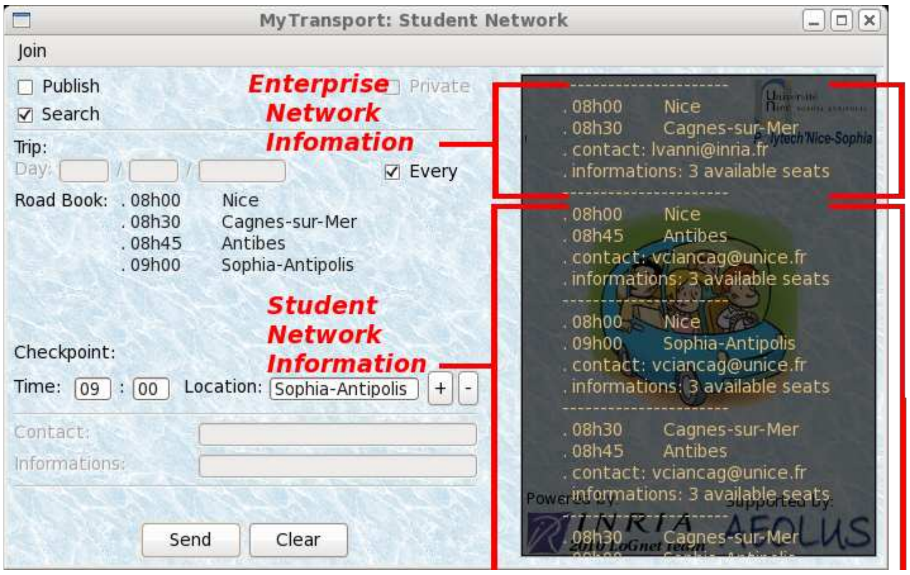


Figure 5.8: Synapse creation

real-world need, such as providing a scalable infrastructure for a community-driven carpool service without requiring a centralized client-server infrastructure.

In this scenario, the idea of interconnecting multiple overlays is applied in allowing individual communities to independently manage their own data (in the form


MyTransport: Student Network
Join
Enterprise Network Infomation
Student Network Information
Send
Clear
Powered by
Supported by:
INRIA
2010 LoGnet team
AEOLUS
Polytech'Nice-Sophia

☐ Publish
☑ Search
☐ Private

Trip:
Day: / /  ☑ Every

```
Road Book: . 08h00    Nice
           . 08h30    Cagnes-sur-Mer
           . 08h45    Antibes
           . 09h00    Sophia-Antipolis
```

Checkpoint:
Time: 09 : 00 Location: Sophia-Antipolis + -

Contact:
Informations:

```
-----------------------
. 08h00     Nice
. 08h30     Cagnes-sur-Mer
. contact: lvanni@inria.fr
. informations: 3 available seats
-----------------------
. 08h00     Nice
. 08h45     Antibes
. contact: vciancag@unice.fr
. informations: 3 available seats
-----------------------
. 08h00     Nice
. 09h00     Sophia-Antipolis
. contact: vciancag@unice.fr
. informations: 3 available seats
-----------------------
. 08h30     Cagnes-sur-Mer
. 08h45     Antibes
. contact: vciancag@unice.fr
. informations: 3 available seats
-----------------------
. 08h30     Cagnes-sur-Mer
```

Figure 5.9: CarPal Students accessing result from Enterprise Network

of published routes), while allowing nearby communities to share routes increasing the chance for users to find a suitable car ride.

There are several improvements such an application could benefit from: for example the integration with existing databases, in order to further extend the search space to canonical web-based communities.

Another evolution in the system could come from the use of an overlay protocol more specialized with the kind of data managed in the application. The adoption of a semantic hash function (such as [Salakhutdinov 2009]) would allow for clustering of semantically close information (i.e. trips heading to sibling destinations or taking place in the same time window) in nearby peers. With such hashing in place the adoption of an overlay protocol more suited to range queries (like P-Ring [Crainiceanu 2007], P-Grid [Aberer 2003] or Skipnet [Harvey 2003]) might lead to semantically significant range queries, where, for example, departure and arrivals can be geographically mapped and queried with a certain range in Km.

Finally, one last possible improvement would be to use a DHT protocol more suited for geo-located information. CAN [Ratnasamy 2001]) 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 [Chawathe 2005]) could ease the query routing and eventually provide some strategic points to place synapsing nodes.

The contributions of this chapter have been published as *CarPal: Interconnecting Overlay Networks for a Community-Driven Shared Mobility*, in Proceedings of Trustworthy Global Computing Conference 2010, Springer LNCS [Ciancaglini 2010].

Chapter 6

# A distributed digital archive for cultural heritage

**Contents**

In this chapter, we present another proof-of-concept for an application relying on an interconnected architecture to provide a new form of interoperability in a real world scenario. Cultural heritage archives all over the world are a typical example scenario that could vastly benefit from interoperability and information exchange. Unfortunately, the lack of standards adoption and general inability of co-operation between different institutions makes it very hard to collaborate. By leveraging the Synapse Framework presented in Chapters 2 and 3, we can allow for the interconnection of different overlay networks, each of them representing the abstraction of a “community” of virtual providers. Data storage and data retrieval from different kind of content providers (i.e.libraries, archives, museums, universities, research centers, etc.) can be stored inside one catalog. We take into consideration the specific case of Serbia’s cultural heritage catalog, and build a system where, while ownership of the content remains within the boundaries of each institution, all the related meta-data can be shared in several distributed overlays, each one communicating with one another.

## 6.1 Context

Digitization is an important step aimed in preservation and promotion of heritage. It safeguards cultural diversity in the global environment and offers a rich treasure to the world-wide public of the Web. Usually, digitization can be seen as a collection of activities, including digital capture, transformation from analogue to digital form, description and representation of heritage objects and documentation about them, processing, presentation and long-term preservation of digitized content, etc.

The document [une 2005] "Recommendations for coordination of digitization of cultural heritage in South-Eastern Europe", accepted at the South-Eastern Europe regional meeting on digitization of cultural heritage (Ohrid, Macedonia, 17-20 March 2005) states that current digitization practice in SEE is still not matching the priorities communicated on the EU-level and that the rich cultural content of the region is still underrepresented in the electronic space. One of the main principles accepted by the participants of the Meeting states that "It is recognized that knowledge of the cultural and scientific heritage is essential for taking decisions concerning its digitization and for interpreting the digitized resources. For this reason, inventorying and cataloging should precede or accompany the digitization of cultural and scientific assets."

At the moment, there is no widespread meta-data standard for describing digitized heritage in Serbia. Actually, although most of the institutions caring about national heritage have started the digitization process, there is no meta-data standard formally accepted at the state level. Because of that we are faced with something that can be called *the meta-data problem*. Different providers of heritage resources (libraries, museums, archives, some research institutions) use international standards appropriate for their specific fields, or ad-hoc methods, or old procedures for describing cultural assets in classical format (formulated in 1980s or early 1990s). In fact, some providers are still waiting for some solution of the meta-data problem and do not do anything related to digital cataloging. It means that digital catalogs in Serbia, if exist at all, cannot help in communication between different kinds of providers and users.

At the international level, there are plenty of meta-data standards for describing heritage resources, for example: Dublin Core [DC ], EAD [EAD ], MARC [MAR ], TEL AP [TEL ], FRBR [web 2007, frb 1998] etc.

Given all of the aforementioned, the Committee for digitization of the UNESCO commission of Serbia has recognized the meta-data problem as the most sophisticated one in the cataloging phase of digitization. During the past years, some efforts were made in the field of standardization, which resulted in the development of the Recommendation for the meta-data format for describing digitized heritage [Z. Ognjanović 2009], but this recommendation has not, still, been accepted as a formal national standard.

There were also some efforts directed towards developing technology for storing these meta-data documents, but there is still no widespread application.

Recent attempts to create digital repositories, such as, for example, Europeana [eur ], are mostly based on centralized architectures. Here we will consider an alternative, decentralized approach, based on overlay networks.

Overlay networks have recently been identified as a promising model to cope with the Internet issues of today, such as scalability, resource discovery, failure recovery, routing efficiency, and, in particular in the context of information retrieval. Many disparate overlay networks may not only simultaneously co-exist in the Internet, but can also compete for the same resources on shared nodes and underlying network links. This can provide an opportunity to collect data on various kind of digitized documents which are, by their nature, highly distributed resources, while keeping backward compatibility, efficient searching, failure resistance, etc.

One of the problems of the overlay networking area is how heterogeneous overlay networks may *interact* and *cooperate* with each other. Overlay networks are heterogeneous, and basically unable to co-operate with each other in an effortless way, without merging, an operation which is very costly since it is not scalable and not suitable in many cases for security reasons. However, in many situations, distinct overlay networks could take advantage of co-operating 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.

More generally, in the context of large scale information retrieval, several overlays may want to offer an aggregation of their information/data to their potential common users without losing control of 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.

The solution could be found in using a meta-protocol which allows a request to be routed through multiple overlays, where one overlay contains one kind of institutions, even using different routing algorithms, thus increasing the success rate of every request.

The ready-to-market DHT(Distributed Hash Tables)-based technology of structured overlay networks is enriched with the new capability of crossing different overlays through *co-located nodes*, i.e.by peers who are, by user's choice, member of several overlays. Such nodes are themselves able not only to query multiple overlays in order to find a match, but also to replicate requests passing through them from one network to another and to collect the multiple results.

## 6.2 Application principles

One of the main features of a distributed catalog is to assist researchers and members of the wider community in retrieving information concerning some fact of interest to them, information which can be provided from different kinds of sources. As mentioned before, digitized documents, by their nature, are highly distributed resources. By connecting different kinds of data providers into one system, we can increase the quality of the resulting information.

In the present work, we consider a distributed catalog which contains only metadata on digital documents which follows a part of the Recommendation for the metadata format for describing digitized heritage, described in [Z. Ognjanović 2009]. One of the main reasons for this is the intellectual property rights issue. Simply, some institutions do not wish to outsource control over their digital repositories, and, instead, choose only to publish the information that they are in possession of a certain document. The digital documents themselves can be retrieved with one of the meta-data fields which contains information on their actual remote location.

A user can connect to one or more communities of which he is member (i. e. he has been invited to or his request has been accepted). Two operations are then available, namely (i) storing a new record and (ii) finding a record which contains some information.

Suppose that we wish to store the following information on one digital object:

```
<digitalObject>
  <title>Title</title>
  <creator>Name</creator>
  <location>link</location>
  <relatedAsset>Related realife object</relatedAsset>
  <note>
    <src lang ="language of the value">value</src>
  </note>
  <archivalDate>date</archivalDate>
  <mimeFormat>mime type</mimeFormat>
  <digitalObjectOwner>Owner</digitalObjectOwner>
</digitalObject>
```

If we were to decide to make the catalog searchable for the values in the fields: *title*, *creator*, *relatedAsset mimeFormat* and *digitalObjectOwner*, then we would store segments in accordance with Table 6.1. More precisely:

| No. | Key | Value |
|---|---|---|
| 1 | $title\sharp Title$ | $hash(✠)$ |
| 2 | $creator\sharp Name$ | $hash(✠)$ |
| 3 | $relatedAsset\sharp RelatedRealifeObject$ | $hash(✠)$ |
| 4 | $mimeFormat\sharp mimeType$ | $hash(✠)$ |
| 5 | $digitalObjectOwner\sharp Owner$ | $hash(✠)$ |
| 6 | $hash(✠)$ | ✠ |

where ✠ represents the full meta-data record on one digital document

Table 6.1: Different data structures stored in the distributed catalog DHT for each entry

1. For every field of a meta-data record which we choose to be searchable, we store the hashed value for the current overlay of the entire meta-data record as value with the key which contains information about the field and its value. Rows 1 to 5 in table 6.1.

2. We store the entire meta-data record as a value with the corresponding key that contains its hashed value for the current overlay. Row 6 in table 6.1.

Note that all of the keys are stored with their hashed values. With this in place, the search mechanism has two phases. During the first phase, we attempt to find the hashed value of the meta-data record (the first kind of entries) and then, during the second phase, to find the entire meta-data record (the second kind of entries) only in the overlays which contain the first kind of entries. Although we have multiple copies of data, so as to accomplish failure resistance of the system, the storage space is of the same complexity as for a standard DBMS with indices. If $N$ and $M$ are the number of overlays and the number of nodes per overlay, respectively, then the time complexity of a search, in the worst case, is $O((N + 1) * (\textit{time to search an overlay with } M \textit{ nodes}))$.

## 6.3 Case study

Institutions which are interested in sharing meta-data information on their digital documents can be connected in different overlays by their nature. So, all archives may be part of one overlay, all libraries of the other, and similarly with museums, research centers, universities, etc. These overlays can be connected by institutions which contain various kind of content, like research centers with important libraries or research centers which are part of the universities, etc. All of these institutions will run the same application.

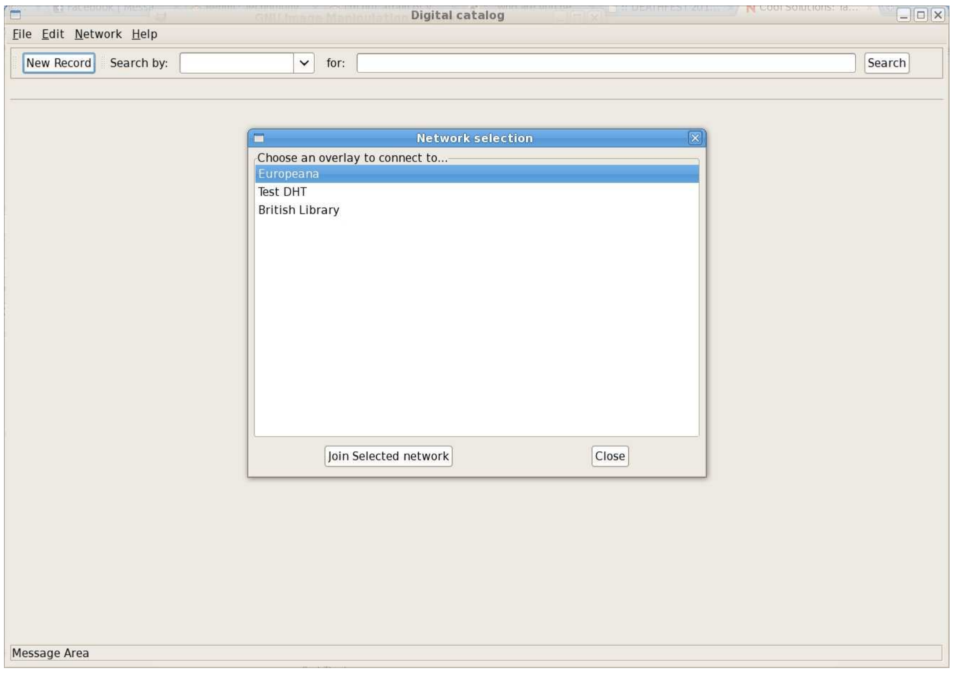


Figure 6.1: Connecting to an overlay

The following proof-of-concept is a simple application to store and retrieve records from one or multiple overlays. It offers the following three functionalities, arranged in a Graphical User Interface developed in Java, for cross-platform compatibility:

- **Join** of a new network.
- **Store** of a new record.
- **Search** for records.

The application is designed using a tabbed organization of different forms. This is to allow the user to easily perform multiple operations at the same time (e.g.doing multiple queries and comparing the results). Furthermore, it constitutes a familiar usage environment, resembling, in the approach, most of modern Internet browsers (multiple tabs, address/search on a top bar). Basic editing features, like saving and loading a record to and from an XML file, copying/pasting and printing the XML raw data, are provided.

### 6.3.1 Network join

As shown in Figure 6.1, upon starting, the program will propose to the user a list of known DHTs to connect to. These represent existing overlays put in place using the same system, which are, therefore, compatible with our software. It is important to notice that, after having connected to a first overlay, a user can choose to further join other available networks. This can be done via the menu entry Network $\rightarrow$ Join, which will propose the same dialog box as in Figure 6.1.

Once being a member of multiple overlays, not only will it be possible to query all of the overlays simultaneously, but, thanks to the capabilities of the synapse protocol described in Chapter 3, it will also be possible to act as a relay, replicating requests from one overlay to another.

### 6.3.2 Storing a new record

Figure 6.2 shows the insertion form for a new record in the DHT.

In this catalog we will store the records which follow a part of the mentioned recommendation of the meta-data format:

- **Title** of the digital document (i.e.electronic book)
- **Name of the author** who made electronic version
- **Link** of the remote location of the digital document
- **Related object** (i.e.hard copy book)
- **Note** or a short description
- **Date** when electronic copy was made

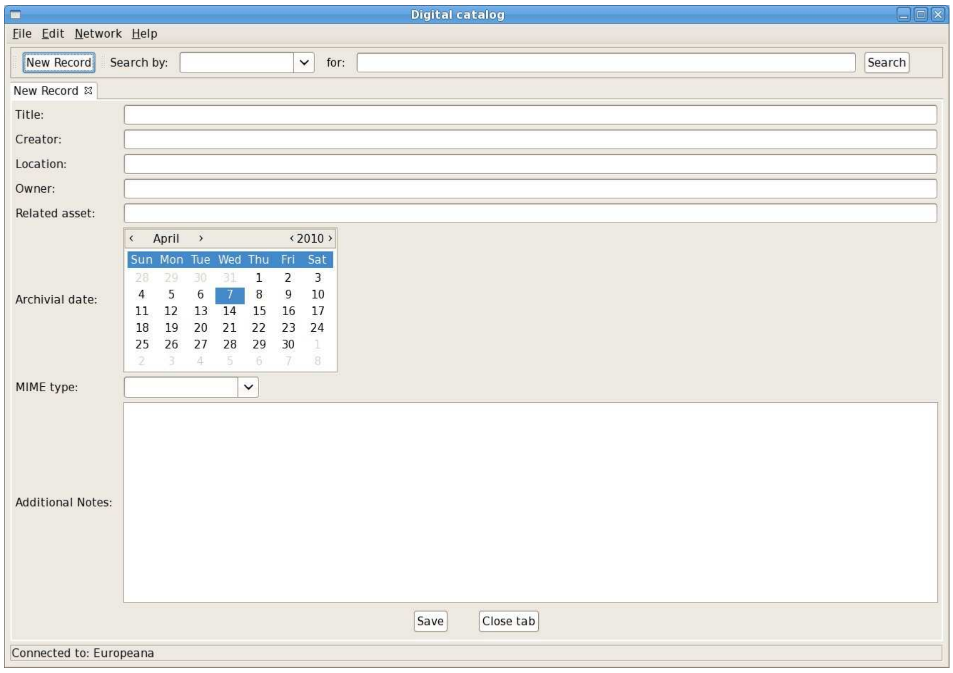


Figure 6.2: Record insertion form

- **Mime type** (i.e.pdf)
- **Owner** of the digital object

While some of the fields may be optional, the ones used as search criteria have to be filled before the record can be saved. Therefore, the "Save" button remains disabled until all of the appropriate text boxes are filled.

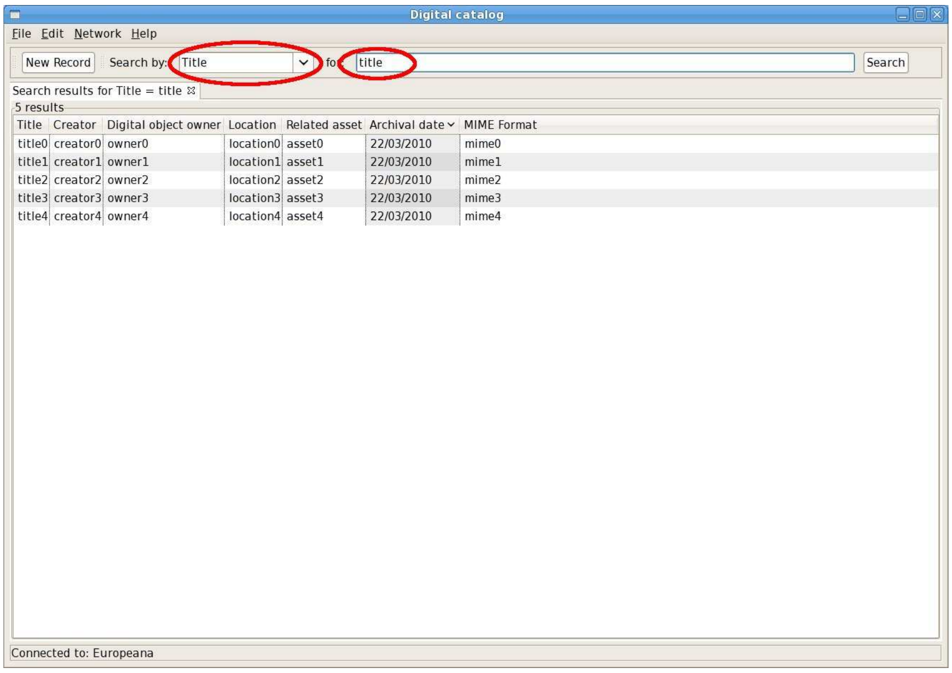
### 6.3.3 Record search

Looking for a record takes place in a way resembling the behavior of most modern Internet browsers: as one can see in Figure 6.3, the search type and field are in the upper toolbar. Here the user can choose the type of search to perform (title, author, owner, related object, mime type) and insert the search key. By pressing the "Search" button, a query for the corresponding key is performed in the overlay (or overlays, if synapses are present or the software is connected to multiple networks). A result summary is displayed in a new tab once the query is over, containing the number of records found and a table with all the records.

To display the details of a record the user can double-click on the corresponding row in the table. This (Figure 6.4) will open a new tab containing record details.

The details tab is similar to the new record form, except that the text fields cannot be edited (although it is still possible to select and copy the text inside).

The button "Display raw XML" will open a new dialog showing the actual XML data.



Search by: Title for: title

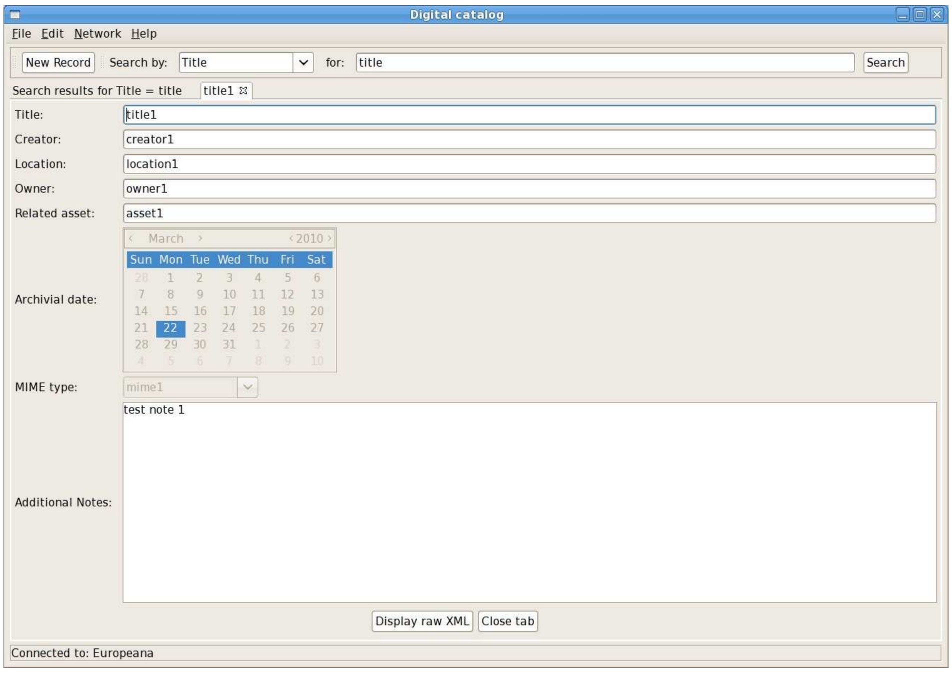
Search results for Title = title

5 results

| Title | Creator | Digital object owner | Location | Related asset | Archival date | MIME Format |
|---|---|---|---|---|---|---|
| title0 | creator0 | owner0 | location0 | asset0 | 22/03/2010 | mime0 |
| title1 | creator1 | owner1 | location1 | asset1 | 22/03/2010 | mime1 |
| title2 | creator2 | owner2 | location2 | asset2 | 22/03/2010 | mime2 |
| title3 | creator3 | owner3 | location3 | asset3 | 22/03/2010 | mime3 |
| title4 | creator4 | owner4 | location4 | asset4 | 22/03/2010 | mime4 |

Connected to: Europeana

Figure 6.3: Search results for a query



Search by: Title for: title

Search results for Title = title | title1

| | |
|---|---|
| Title: | title1 |
| Creator: | creator1 |
| Location: | location1 |
| Owner: | owner1 |
| Related asset: | asset1 |
| Archivial date: | March 2010, 22 |
| MIME type: | mime1 |
| Additional Notes: | test note 1 |

Connected to: Europeana

Figure 6.4: Details of a retrieved record

## 6.4 Conclusion

In this Chapter we have shown that the Synapse Framework has good potential as yet another new concept of DBMS, depicting its applicability to a real-life situation.

As mentioned before, within this system we can also store the digital documents themselves. We have also decided that in the current phase, this should be out of scope but we consider this to be a possible system improvement with great potential.

As a positive side-effect, we believe that our catalog can lay promising groundwork for a low-cost solution to cultural interconnection of the institutions inside the Balkan region.

The contributions of this chapter have been published as *A Distributed Catalog for Digitized Cultural Heritage*, in Proceedings of ICT Innovations 2010, Springer LNCS [Marinković 2011].

Part III

# Beyond overlays: content-based routing for real-time video streaming

Chapter 7

# Content based enhancements in P2P-TV: promises and drawbacks

**Contents**

Traditional gossip-based P2P-TV systems are broadly recognized as effective and scalable solutions for Internet real-time video delivery. Nonetheless, they can incur significant performance impairments in highly heterogeneous environments, due to content and bandwidth bottleneck issues. In fact, because of topological constraints of the overlay, a peer might not be able to locate or retrieve the required chunk within the playout deadline because of a limited set of neighbors or insufficient bandwidth. In an attempt to resolve this challenging issue, content based approaches could be exploited because of their inherent ability to straightly locate and retrieve a specific information within a distributed system. Hence, in order to understand to what extent such a solution could improve P2P-TV, we analyze herein an extension of the traditional gossip-based approach by means of an optimized content-based retrieval protocol derived from Distributed Hash Tables (DHT). The solution we propose is fully backward compatible and can sit on top of existing gossip based protocols so that it could be selectively turned on and off based on the specific application scenario. Extensive simulations showed initially that a plain DHT-like overlay cannot bring a notable improvement on the chunk loss when real time constraint are enforced (i.e. low playout delay), however, with appropriate optimizations, performance improvements up to 20% could be reached. Furthermore, the system proved to be consistent and robust even under stressful conditions, with heavy network load and high churn rates, and shows interesting properties regarding the peers' upload bandwidth exploitation.

## 7.1 Introduction and Related work

### 7.1.1 Gossip-based protocols

In a gossip-based peer-to-peer (P2P) communication system, all nodes (peers) interested in the same common content cooperatively build up a high-level overlay network that, by exploiting the peers' upload bandwidth, allows for scalable and fast services to be provided [Ceballos 2006, Pouwelse 2005]. In particular, each peer establishes links with a limited set of neighbors, representing the nodes at one-hop distance in the overlay topology, and exchanges pieces of data with them, receiving content from multiple sources, while serving multiple neighbors. Every peer offers its own upload bandwidth for content distribution, thus eliminating the need for high-capacity servers. This advantage has been fruitfully exploited in the past to design powerful file transfer applications [Liu 2009].

Recently, the attention of the scientific community and industry has turned towards P2P-TV, due to the prevalence of this field in everyday life, as well as in the market [Li 2006, Ali 2006]. In P2P-TV systems, a multimedia data source generates a series of *chunks*, with each one of them containing a part of the audio/video bitstream, and makes them available to all of the peers connected to the overlay [Liu 2008].

The main difference with respect to file sharing applications is related to the strict delay requirements of P2P-TV services, which impose a deadline on each chunk at generation time: when a chunk is received after its *playout delay* (the deadline) has elapsed, it is considered lost [Xiao 2008]. Increasing the playout delay allows for more and more chunks to be received within this deadline: hovewer, this advantage comes at the expense of the Quality of Experience (QoE) of the users, which is very sensitive to the timeliness of TV services [Leonardi 2008].

Other performance limitations of classic P2P-TV are related to the so called bandwidth and content bottlenecks [Ciullo 2010]. It is, in fact, possible that a chunk whose deadline is going to expire cannot be downloaded by a given node because none of its neighbors have that chunk (content bottleneck) or the upload bandwidth of the neighborhood is insufficient (bandwidth bottleneck). In both of these cases, the chunk in question would be lost, and, consequently, the QoE would be impaired. Increasing the degree of cooperation of the overlay network by allowing each peer to establish direct links with many other peers is not a viable solution because of the overhead required to handle the high number of connections in each neighborhood.

### 7.1.2 Content adressable networks

Content-driven routing has become more and more popular in the last few years, and there have been even propositions such as [Jacobson 2009a] to use it as an alternative to classic IP routing.

Structured overlays such as Chord [Stoica 2003], Kadmelia [Maymounkov 2002b] or CAN [Ratnasamy 2001] use content-driven routing to provide a scalable lookup mechanism of data in the form of key-value pairs. Most of them uses hash functions in order to map node IDs and content keys to a common adressing space. Because of this they are commonly known as *Distributed Hash Tables*. Routing of a request on a structured overlay follows a path usually dependent only on the key and the node routing table. As a consequence, every request usually follows a different path to its destination, thus providing load balancing across the overlay.

To the best of our knowledge, very few works have concentrated on the use of content addressable networks in P2P real-time video streaming, either with different purpose to ours or under unrealistic assumptions. The open source P2P-TV client Goalbit [Bertinat 2009] integrates a KAD implementation. But in this case, as in many other cases, DHT is not used for chunk delivery, but only for discovering the list of peers participating in the stream, as a distributed tracker solution. As a consequence, the DHT does not influence streaming performance. In [Yiu 2007] DHT are used for Video streaming, but for on Demand applications. A VoD streaming system shows good performance when peers watching the same portion of the video are neighbors. In case of fast forward or rewind, a peer, to find its neighborhood, makes a request on the DHT for the part of video he wants to stream. In [Castro 2003], the DHT is used to allow the creation of a multiple-tree overlay and to ensure its connectivity. In tree based overlays, in fact, churning events can easily lead to a disconnected overlay graph, isolating a group of peers from the streaming. The DHT here serves to minimize the probability for each peer to be completely disconnected, and to speed up the recovery of the original graph.

In some other works, as in [Jeonghun 2008] and [Nguyen 2008], the whole chunk delivery is done by means of a DHT, but no explicit evidence of the ability to support Internet TV is provided, i.e., chunk losses and communication delays have not been evaluated.

The performance evaluation of the proposed approaches does not take into account the timeliness of the system, but only quantifies system performance in terms of successful DHT lookups, and in terms of average number of hops needed to find resources. However, especially in real-time video streaming, a critical aspect is evaluating the chunk communication delay for each peer, as done in our work. We try to have an evaluation of chunk delivery which is as realistic as possible, by taking into account i) the latencies with which chunks and signalling messages are sent, ii) the variable transmission delay experienced by chunks according to the bandwidth availability in a heterogeneous scenario and iii) the queue delay related to multiple requests received by the sender peer.

### 7.1.3 Hybrid delivery algorithms

To overcome the content and bandwidth bottleneck issues exposed in Section 7.1.1, a pull mechanism based on content-based requests similar to the ones performed in DHTs could be put in place to target peers outside the gossip neighborhood and retrieve chunks yet not available. To this subject there have been few works in the literature, notably [Locher 2007] and [Shen 2010a]. Authors in [Locher 2007] propose a whole new chunk diffusion protocol, rather than a simple enhancement of existing methods, that can leverage both a push and a pull based mechanism, while authors in [Shen 2010a] implemented an algorithm similar to the one analyzed here but without our proposed optimizations. As the results in Section 7.3 will show, a simple DHT mechanism cannot scale properly when real-time constraints are enforced.

It is worth to note that this kind of hybrid gossip and DHT based approach has been previously proposed in literature also for applications other than video streaming [Zaharia 2006], [Luo 2008]. These kind of applications, however, do not need to respect strict time constraints for content delivery, and do not need to speed up the resource retrieval time employed by DHT. On the contrary, in file sharing, information retrieval, they take into account the data integrity, preferring packet loss avoidance more than limited download delay.

For that purpose, we designed a content-based pull mechanism, henceforth referred to as HyDeA, and analyze the performance gain it can provide when integrated in a classic P2P-TV system, while maintaining backward compatibility and low signaling overhead. To achieve this, several optimizations had to be implemented over a canonical DHT routing scheme, in order to better exploit the system's network heterogeneity and reduce the routing complexity, as detailed in Section 7.2.

In particular, we introduce a *pseudo-cache* and a *bandwidth selective peer join* mechanism in order to, respectively, (*i*) increase the retrieval efficiency of the system and (*ii*) fully exploit the capabilities of high-bandwidth peers. The *pseudo-cache*, described in detail in Section 7.2, allows a peer receiving a chunk request to send the content (if present in its local storage) back to the requesting peer, thus shortening communication delays. The probability of finding middle peers holding the desired chunk is high, considering that every chunk should be ideally received by every peer participating in the stream. Our algorithm, finally, allows chunks to be retrieved only from high-performance peers, i.e., on peers with an available upload bandwidth greater or equal to the video rate. With this consideration, chunks requested to the content-based overlay are retrieved faster, because of the high bandwidth of sender peers. Furthermore, the routing complexity for requests to reach the assigned peer is $O(log(N_{HP}))$, where $N_{HP}$ is the number of high-performance peers, and not the total number of peers participating in the stream. These choices we have made have great impact in real time scenarios.

Our delivering system is compared with a simple hybrid gossip+DHT- based chunk delivering, and extensively analyzed using simulations, in conditions as realistic as possible, considering peers' heterogeneity, using a real video model and taking into account network latencies and signaling messages. Simulations results in Section 7.3 not only show that our system outperforms the simple gossip-based ones in terms of chunk losses, but also demonstrate that using a simple DHT on top of a classic gossip-based P2P-TV systems does not help improving the performance. Moreover, interesting effects regarding the peers' bandwidth exploitation are shown, that could potentially be exploited in mobile scenarios.

## 7.2 System description

As said above, HyDeA adds a *per-request* mechanism, based on content-addressable requests, on top of a classic gossip-based delivery. The main chunk diffusion follows the canonical gossip-based mechanism, but, in parallel, each peer can ask the structured overlay, in defined intervals, for an expiring chunk that has not been delivered yet by the gossip. The decoding buffer is common to the two mechanisms, so there is no distinction between chunks received via the gossip overlay or the structured one, and every chunk received via one channel will be available to the other, i.e. every chunk received via the gossip overlay is available for any ongoing content-driven request. This section presents the details of the overlay used for content-addressable chunk requests as well as the chunk distribution algorithm for both gossip- and content-addressable methods.

### 7.2.1 High Performance DHT overlay

To be able to perform specific requests for a video chunk beyond the gossip neighborhood, peers form a High Performance DHT (henceforth referenced as HP-DHT) beside the existing gossip one. This overlay relies on a protocol based on DHT routing, without the data storage semantics and with strong optimizations in order to reduce the query response time (asynchronous non-blocking messaging, amongst others).

As described in [.Shen 2009], a DHT works by mapping peers and content on a common addressing space. The mapping is usually done by means of a hash function, which guarantees a uniform pseudo-random distribution of both peers and contents in that space. Every peer is responsible for a certain interval in the space, according to various proximity metrics (e.g., linear distance as in Chord [Stoica 2003], XOR metric like in Kademlia [Maymounkov 2002b] or geometric distance as in CAN [Ratnasamy 2001]), and will, therefore, keep every content whose mapping falls into said interval. Data is accessed via keys, by sending a data-lookup message that is routed through the overlay to reach the node which is responsible for a given key. Most DHTs guarantee that lookup has logarithmic complexity, i.e. $O(log(N))$, where $N$ is the number of all nodes participating in the DHT.

The HP-DHT overlay uses a Chord-like ring topology where every peer $P$ has a logical identifier $ID_P$, derived by hashing its IP address and port, and a set of pointers to other peers in the overlay to be used as routing tables, namely:

- a list of $k$ pointers to the peers whose $ID$ directly follows its $ID$ (i.e. the *successors*);
- a pointer to the peer with the biggest $ID \leq ID_P$ (i.e. the *predecessor*);
- a routing table, containing for each entry $i$, a pointer to the peer responsible for the key equal to $ID_P + 2^i$ (i.e. the *finger table*).

These data structures are used to route a request to the responsible peer, and are filled upon peer join and refreshed periodically with new peers who might have joined the overlay.

### 7.2.2 Bandwidth selective peer join

To keep HP-DHT performance high, when a new peer joins the overlay, its presence will be made known to the rest of the overlay only if its declared upload bandwidth is higher than a given *minimum bandwidth threshold*, an approach similar to what has been done in [Brampton 2006] for a Pastry DHT.

This distinction categorizes HP-DHT nodes in two classes:

1. peers with an upload bandwidth greater than the minimum bandwidth threshold (HP peers) will become active members of the HP-DHT overlay, routing lookup messages and serving video chunks upon request;
2. peers with an upload bandwidth less than the minimum bandwidth threshold (lightweight nodes) will just update their own routing tables with the appropriate HP peers, to be able to send request messages, but will never serve video chunks nor route lookup messages.

As it will be shown in Section 7.3, this approach helps circumvent bandwidth bottlenecks that affect gossip-based systems in the following ways:

- it avoids having slow peers serving video chunks close to expiration;
- it speeds up lookup requests by lowering the routing path from $O(log(N))$ to $O(log(N_{HP}))$, where $N_{HP} \leq N$ is the number of HP peers, and avoiding having slow peers along the path;

- it helps saturate the HP peers' highly unexploited upload bandwidth, by concentrating part of the load on them.

In order to achieve good performances rather than overloading the overlay, a minimum bandwidth threshold value at least equal to the video rate should be chosen.

### 7.2.3 Chunk retrieval

To receive chunks, a peer adopts two different mechanisms:

1. following its gossip protocol, with a relatively low latency, it receives chunks from its neighbors after `OFFER/SELECT` exchanges. In particular, as described in [Fortuna 2010], every peer periodically receives some `OFFER` messages from its neighbors, according to the neighbors' bandwidth availability. The `OFFER` message, containing the neighbors buffer map, indicates that the peer can select and retrieve one of the offered chunks, according to the chosen chunk scheduler. We use a random chunk scheduler, since it has been demonstrated in [Fortuna 2010] to respond more robustly than the others to variations of system parameters. In this way every peer requests and quickly obtain in in a gossip-based manner the bigger part of video chunks;

2. every `REQ_INTERVAL` seconds, a peer can perform an explicit request to the HP-DHT for a chunk amongst those not yet received from its neighbors. Such a request consists of a lookup `GET` message containing the hashed chunk number. The destination peer receiving said message then initiate a specific `OFFER/SELECT` exchange with the requesting peer. The chunk to request is selected within a moving request window, i.e. a subset of the decoding buffer.

### 7.2.4 HP-DHT pseudo-cache

To further reduce the logarithmic cost of each lookup request in the HP-DHT, each peer along the routing path checks for the requested chunk ID in its own decoding buffer, and, in case it is present and the peer's transmission queue is free, it responds to the request itself. Thanks to this "pseudo cache" mechanism, a `GET` request likely takes less than $log(N_{HP})$ hops to find a destination. Content based chunk requests to an HP peer $P_i$ are served so that the upload bandwidth dedicated to serving the HP-DHT is, at maximum, $Up_{HP} = \frac{Up_i}{d_i + 1}$, where $Up_i$ is $P_i$'s upload bandwidth and $d_i$ its connection degree, i.e., the number of neighbors. This way, the anonymous content-based neighbor is treated exactly as one extra gossip-neighbor. To be able to answer a chunk request, the peer makes sure that the time requested to void its transmission queue and the time to send the chunk itself will not be higher than the chunk deadline, i.e.

$$C_d > t_{now} + (\frac{Size_Q + Size_C}{Up_{HP}})$$

where $C_d$ in the deadline of the requested chunk, $t_{now}$ is the actual instant, $Size_Q$ and $Size_C$ are, respectively, the size of the transmission queue for chunks requested via the HP-DHT and the size of the requested chunk, and $Up_{HP}$ is the upload bandwidth reserved for HP-DHT deliveries.

### 7.2.5 Chunk seeding

Whenever a new chunk is generated, the broadcasting peer performs two different kind of seed in the network:

1. the chunk is first sent randomly to one of the source's neighbors, according to the gossip protocol;

2. the chunk is also explicitly sent to the responsible peer according to the content-based routing, in order to make it immediately available to HP-DHT requests. The peer is selected by sending a `FIND` message on the HP-DHT containing the chunk's hashed ID, as if it was a lookup request. This time though, the pseudo-cache cannot be taken into account, and the routing has a complexity of $O(log(N_{HP}))$.

Despite what was intially expected, the chunk seeding on the HP-DHT alone didn't show a contribution on the overall system performance (being an additional seeding on the overlay), but it had to be kept in order to guarantee the success of content-based requests (who would otherwise be subject to the chunk's successfull reception by the responsible peer, a condition the gossip protocol cannot ensure).

## 7.3 Performance analysis

We herein present the results of our simulations for the system described above. To better understand the effectiveness of our optimizations, we choose to compare HyDeA with both a simple gossip-based system, and a hybrid gossip+DHT system, in which neither pseudo-cache nor bandwidth-aware choice of DHT peers is included, so as to use the plain gossip as a reference and to analyze the improvements of our HP-DHT under the light of the limitations of a simple gossip and structured overlay network.

### 7.3.1 Settings

In order to keep the simulations as realistic as possible, we decided to take into account the characteristics of a real video, e.g., the musical video clip "Pink" by Aerosmith, at spatial resolution 352x240 and temporal frequency of 25 fps. The number of streamed chunks is 1000, for a 40s long streaming simulation. The sequence has been encoded in h264/AVC, with a GOP structure IDR 7xPBbb.

We avoid generating chunks containing more than one video frame, in order to avoid further delays due to video data aggregation on the source.

For the gossip overlay, a Bittorrent-like system has been chosen, having a mesh topology [Magharei 2007]. Each peer has 70 neighbors directly connected and every time it has free upload bandwidth to serve a chunk, the destination peer is chosen according to its generosity, as described in [Leonardi 2008]. The probability to select one neighbor as destination is, in fact, proportional to its active upload bandwidth. On top of such an overlay we have first implemented a plain DHT, based on the Chord algorithm, and our optimized HP-DHT, that introduces the improvements described in Section 7.2.

We used the simulator developed inside the NAPA-WINE European project, P2PTV-sim.The original simulator is available on line at [nap ], while the source code of our modifications can be retrieved at [log ]. The strength of the NAPA-WINE simulator, as described in [Fortuna 2010], is the integration of a synthetic way to evaluate application-level latencies among peers, which are simulated to be realistically dislocated on the globe, following the statistical studies in [int ]. The simulator has been modified in order to integrate a Chord-based chunk diffusion algorithm, with and without the proposed optimizations.

As frequently done in the literature, the download bandwidth is set to infinite, as it is assumed to be much higher than the video rate, and the bottleneck in video streaming applications is usually caused by the peers upload bandwidth.

We try to evaluate chunk delivery as realistically as possible, taking into account (*i*) the latencies with which chunks and signaling messages are sent, (*ii*) the variable transmission delay experienced by chunks according to bandwidth availability in a heterogeneous scenario, and (*iii*) the queuing delay related to multiple requests received by the same sending peer. Unless specified differently, the default system parameters can be assumed as in Table 7.1.

| **Parameters** | **Values** |
|---|---|
| Playout delay | 10s |
| Number of peers | 1000 |
| Churn rate | 0% |
| Load factor | 1.3 |
| Bandwidth configuration (see Table 7.2) | $E_Q$ |
| Request Interval | 0.1s |
| Request Window Offset | 1.5s |
| Request Window Width | 2s |

Table 7.1: Sistem's default parameters

In what follows, we highlight the behavior of the proposed system against different overlay setups (peers' bandwidth configuration, number of peers, churn rate etc.). We will show that the insertion of HP-DHT brings advantages in all of the analyzed topologies without significant overhead, thanks to the integrated content-driven diffusion mechanism. Furthermore, we will demonstrate that simply integrating a traditional DHT retrieval mechanism does not improve gossip delivery in real-time contexts.

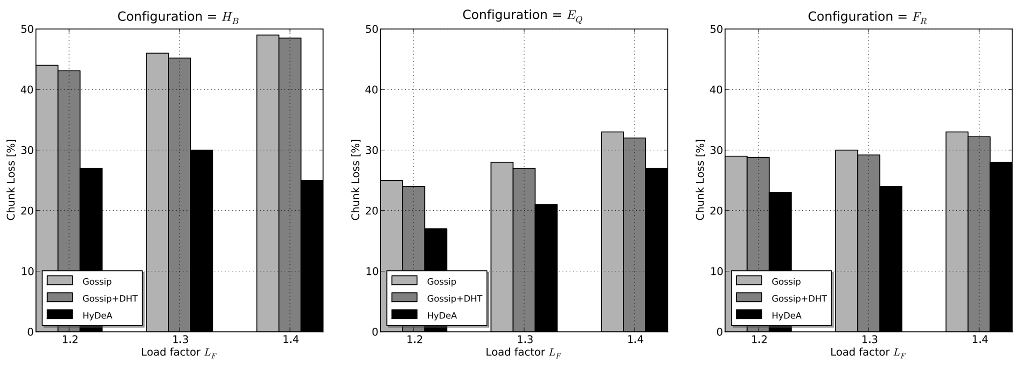


Figure 7.1: Chunk loss on different network configurations

### 7.3.2 HyDeA performance under different load conditions

In this subsection we begin evaluating the proposed system perfornamces in different heterogeneous scenarios with 1000 peers. We considered heterogeneous scenarios made up of peers belonging to different classes, characterized by different upload bandwidths. Peers are divided into four classes, and peers of the same class have the same upload bandwidth. Three different configurations have been used, keeping the same average available bandwidth for all of them, but varying bandwidth repartition amongst classes and the percentage of peers in each class. In order to study the system under stressful conditions, 1.2, 1.3 and 1.4 have been chosen as load factors, obtaining high loss percentages, and evaluating the maximum benefit that a hybrid systems can bring. The load factor is defined as

$$\frac{Video\,Rate}{Average\,Available\,Bandwidth}$$

e.g., an average available bandwidth of 1.1 Mbps across the overlay and a video rate of 1.4 Mbps would give a network load factor of approximately 1.3.

Table 7.2 shows different bandwidth configurations adopted under the aforementioned load factors. For each load factor $L_F$, the 3 configurations are shown $(H_B, E_Q, F_R)$, with both the percentage of peer per each class, and maximum bandwidth for the class. In particular, $H_B$ has a bigger percentage of high bandwidth peers, while $F_R$ has more free riders with no upload bandwidth, and $E_Q$ has the same percentage of both (10%). In all three configurations, the HP peer set is made of Class 1 peers.

In Figure 7.1, the chunk loss percentages for the three configurations are reported. A simple gossip+DHT system gains 1-2 percentage points over the gossip-only system, whereas our HyDeA outperforms them both reaching a consistent gain up to 20%, and not lower than 7%. The gain with the proposed system is present regardless of the load, and is particularly evident in the $H_B$ configuration. That is because in such a configuration, a greater percentage of HP peers is present (15% instead of 10%), thus increasing the capacity of the set of content-driven suppliers.

| | | *Peer classes* | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Class 1 | | Class 2 | | Class 3 | | Class 4 | |
| | | [%] | [Mbps] | [%] | [Mbps] | [%] | [Mbps] | [%] | [Mbps] |
| $L_F$: 1.2 | $H_B$ | 15% | 5.3 | 35% | 0.636 | 40% | 0.371 | 10% | 0 |
| | $E_Q$ | 10% | 5.3 | 40% | 1.06 | 40% | 0.53 | 10% | 0 |
| | $F_R$ | 10% | 5.3 | 30% | 1.272 | 30% | 0.848 | 30% | 0 |
| $L_F$: 1.3 | $H_B$ | 15% | 5 | 35% | 0.6 | 40% | 0.35 | 10% | 0 |
| | $E_Q$ | 10% | 5 | 40% | 1 | 40% | 0.5 | 10% | 0 |
| | $F_R$ | 10% | 5 | 30% | 1.2 | 30% | 0.8 | 30% | 0 |
| $L_F$: 1.4 | $H_B$ | 15% | 4.54 | 35% | 0.545 | 40% | 0.318 | 10% | 0 |
| | $E_Q$ | 10% | 4.54 | 40% | 0.91 | 40% | 0.454 | 10% | 0 |
| | $F_R$ | 10% | 4.54 | 30% | 1.09 | 30% | 0.727 | 30% | 0 |

Table 7.2: Bandwidth distributions under different load factors

On the contrary, in the $F_R$ configuration, the performance of the hybrid systems slightly decreases, either for gossip+DHT or for HyDeA. Here, suppliers dedicate a great part of their upload bandwidth to satisfy requests made by free riders, de facto serving chunks to peers that will never replicate them.

### 7.3.3 Bandwidth exploitation

An important parameter to consider is network bandwidth exploitation. Thanks to the content-driven requests, the excess bandwidth on the Class 1 peers can, in fact, go and benefit directly the Class 4 peers who suffer the highest chunk losses. Figure 7.2 (a), (b), and (c) show the chunk loss per peer (averaged, for the purpose of clarity, over a 10-peer window). Under every load factor it is easy to see how the simple gossip system does not manage to satisfy, due to its tit-for-tat policy, the free riders' requests. On the other hand, as shown in Figs. 7.2 (d), (e) and (f), the upload bandwidth of the best performing peers (averaged for clarity over a 5-peer window) remains unexploited to a great extent, 2-3 Mpbs, depending on the network configuration, against a nominal bandwidth of 5 Mbps.

This is an interesting effect of HyDeA since it acts as a leveling mechanism for the network heterogeneity in the system, and brings improvements to the least performing without affecting the rest, an implication particularly relevant in scenarios (such as mobile P2P) where some peers may suffer from a lack of upload bandwidth.

A simple gossip+DHT system proves to be ineffective in circumventing such bandwidth bottlenecks, whereas HyDeA manages to lower the chunk loss up to 22%, and facilitates exploitation of the HP peers' bandwidth, up to an additional 1 Mpbs (see Figure 7.2 (c)).

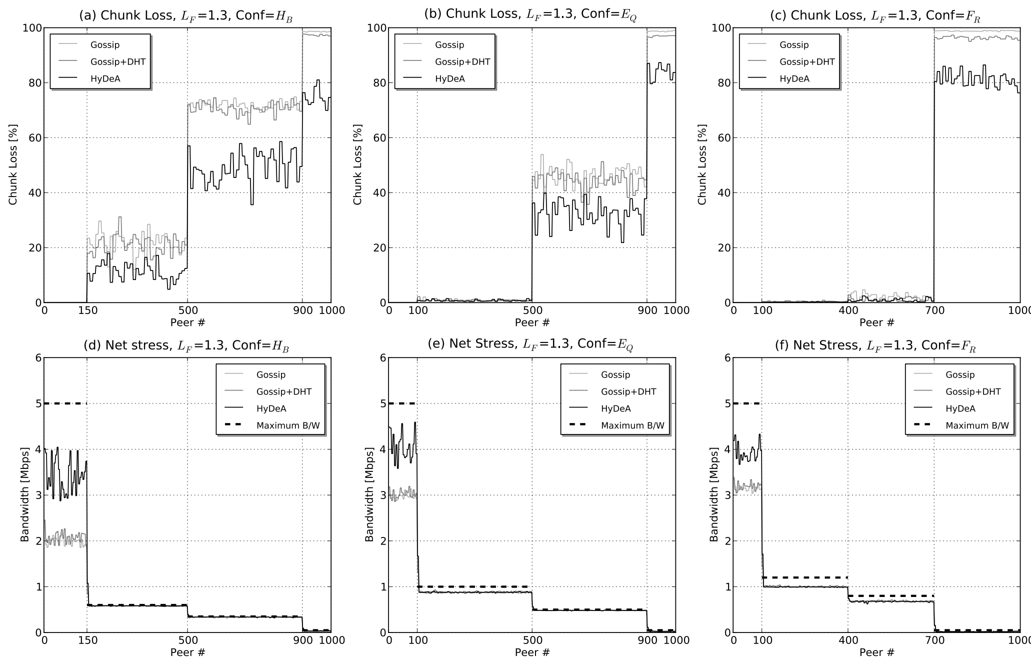


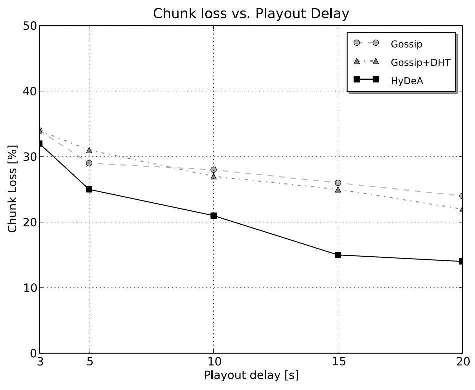
Figure 7.2: Chunk loss per peer (top row) and Bandwidth saturation (bottom row)

### 7.3.4 HyDeA performance at different playout delays

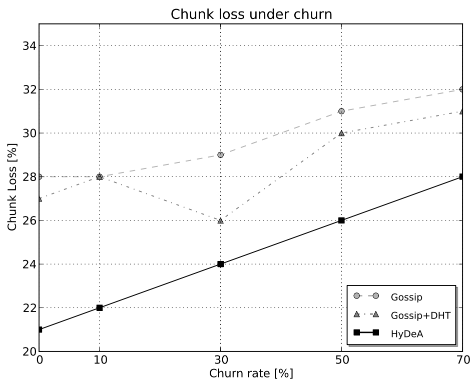
Playout delay is a key parameter in real-time P2P-TV systems. The smaller the playout delay, the more the real-time nature of the system is respected. However, lowering the playout delay also implies making delay constraints in chunk delivering stricter, thus increasing the probability of chunk loss. The optimizations in the proposed system for speeding up chunk delivering are particularly evident in Figure 7.3(a). For every playout delay, HyDeA is able to deliver a greater percentage of chunks on time. The improvements are more evident for high playout delay values. On the contrary, simple gossip+DHT systems introduce improvements only for increasingly high playout delays, thanks to higher delays in chunk diffusion.

### 7.3.5 Robustness to churning

In this subsection, we will focus on the behaviour of the system in presence of churning. We define, as churn percentage, the percentage of peers disconnecting from and reconnecting to the overlay once during the simulation. For instance, when a 70% churn simulation is done in a 1000 peer system, there are 700 disconnection and 700 join events in just 40s of streaming. We chose to study as well system performances with different churning percentage, in stressful conditions. Simulation results reported in Figure 7.3(b) show the robustness of the HyDeA when faced



(a) Chunk loss vs. playout delay



(b) Chunk loss vs. churn rate

Figure 7.3: Chunk loss vs. different parameters

with high-churning scenarios. It is worth mentioning that Chord-like topologies are known to be resistant to churn. Furthermore, HP-DHT implements special precautions to avoid “breaking” the Chord ring due to too many disconnections, by storing several successors for each peer.

Without any optimization (gossip+DHT), the chunk loss percentage has been registered to be 31% in case of 70% churn percentage, only 1 percentage point less than the pure gossip system. However, since mesh-based systems tend to be much more robust to churning than any other structured delivering mechanisms, the gain slightly decreases on increasing churn percentage. Same results were obtained for 2000- and 3000-peer simulations, confirming the scalability of the proposed solution.

### 7.3.6 Signaling overhead evaluation for different HP-DHT parameters

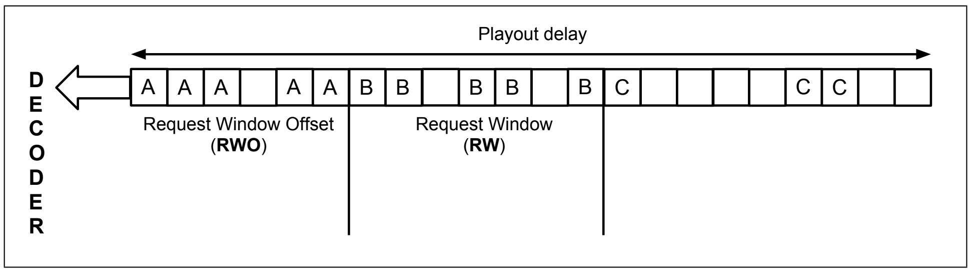


Figure 7.4: HP-DHT chunk selection

In the decoding buffer, we can identify three different kind of missing chunk slots, as shown in Figure 7.4:

- slots $A$ are too close to expiration, and therefore could not be retrieved on time from the HP-DHT. They could, however, still be received from one of the

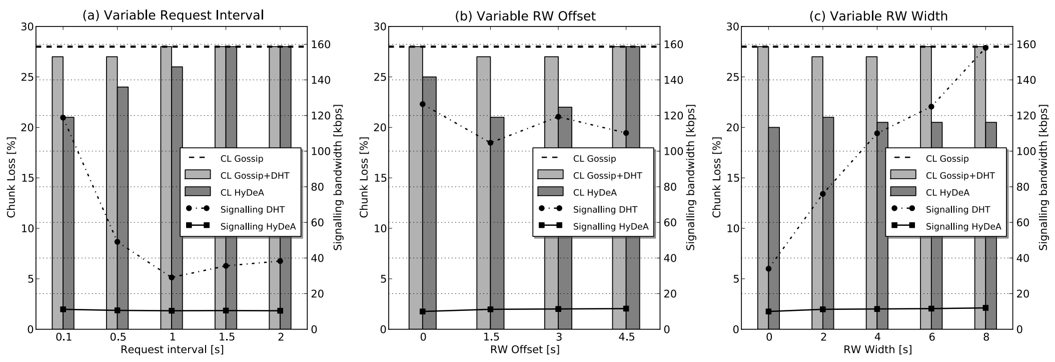


Figure 7.5: Chunk loss and signaling bandwidth for different DHT parameters

neighbors;

- slots $B$, in the Request Window (RW), can be requested from the HP-DHT;
- slots $C$, in the rest of the buffer, are retrieved, like slots $A$, by the gossip protocol only.

In this subsection, we evaluate the performance, either in terms of chunk losses or signaling overhead, while tuning the parameters concerning the HP-DHT Request Window.

#### 7.3.6.1 Request Interval.

The first parameter we would like to tune (denoted by `REQ_INTERVAL`) is the minimum period between two chunk requests to the HP-DHT made by the same peer. By decreasing `REQ_INTERVAL`, the number of requests increases and, consequently, the number of retrieved chunks increases which, in turn, leads to loss minimization. Thanks to the pseudo-cache mechanism, messages tend not to propagate along the whole routing path, and the signaling overhead is kept very low, regardless of the parameter value. On the other hand, a simple DHT tends to propagate request messages across all of the $log(N)$ hops of the routing path, causing a rapid increase in the signaling, when the `REQ_INTERVAL` is set to a low value.

In Figure 7.5 (a), the gossip+DHT signaling overhead is studied against different values of `REQ_INTERVAL`. The average signaling overhead, using a traditional DHT without optimization reaches 7% of the video rate (118.8 Kbps vs. 1.4 Mbps), while the benefit, in terms of reduced chunk loss, does not exceed the 1%. These results show that a traditional DHT integration in real-time gossip based systems does not yield any improvements on its own.

In contrast to this, with the lowest `REQ_INTERVAL` and considering the above video rate, the HP-DHT signaling adds only a 0.8% overhead, while bringing an average of 7% gain on chunk loss. For that reason, we considered safe to keep the lowest parameter as the default one for the other simulations.

#### 7.3.6.2 Request Windows Offset.

The Request Window Offset (denoted by `RWO`) defines the temporal shift of the DHT Request Window (denoted by `RW`) inside the decoding buffer (as shown in Figure 7.4). If a `RWO` = 0 is chosen, the Request Window will contain the oldest chunks, near to their deadline. By increasing the `RWO`, we shift the Request Window towards the most recent chunks. Figure 7.5 (b) shows that a point of optimum for the Window Offset is the medium-to-low value of 1.5s. There are, in fact, two aspects to consider when tuning the `RWO`:

- the DHT Request Window has to be close enough to the end of the decoding buffer in order to include the oldest missing chunks, that are close to expiration and have not been delivered by the gossip neighbors. Missing chunks, close to the deadline, have a high probability of being missing also in the peer's neighborhood, and making a content-driven request could help to overcome the so-called content-bottleneck chunk losses [Ciullo 2010];

- on the other hand, choosing a Window Offset close to 0 means requesting chunks too close to the deadline, i.e. chunks that would probably expire before the request could be satisfied.

In this case, the signaling of both gossip+DHT and HyDeA remains steady for each parameter, since increasing it does not lead to an increase in the number of requests. However, as shown before, the HP-DHT signaling remains lower than the DHT one by a factor of 10.

#### 7.3.6.3 Request Window Width.

The Request Window Width (denoted as `RWW`) is the size, in seconds, of the Request Window. As shown in Figure 7.5 (c), the Request Window Width is not particularly relevant, neither in terms of losses, nor in terms of signaling overhead. That is because the HP-DHT scheduler always gives priority to the oldest chunk in the window. Because of that, even in presence of a wider `RW`, the chunks requested will always be the ones close to the window's end, and due to the limited request interval, increasing the size does not lead to an increased number of requests.

### 7.3.7 Comparison with an adaptive overlay

In this last section we perform a brief comparison against a gossip protocol with a topology optimized overlay [Ren 2008], to see whether the improvements introduced by the content based layer persist. The reference used in this test is a gossip based overlay where each peer can increase or decrease the number of neightbors according to its upload bandwidth. Moreover, the choice of neighbors can be performed either randomly or according to latency criteria. For our tests we decided to have the most part of neighbors (80%) selected according to latency criteria and the rest (20%) chosen randomly, since it showed to be the best performing setup in thsi scenario.

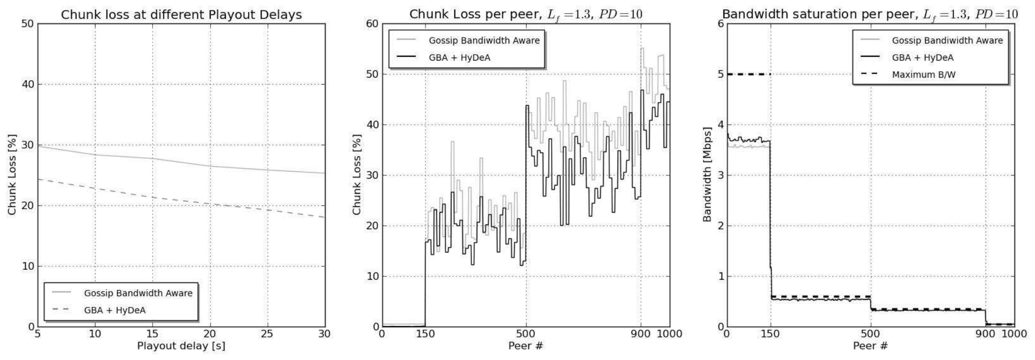


Figure 7.6: Chunk loss at different playout delays (a), Chunk loss per peer (b) and Bandwidth exploitation per peer (c)

Even with an adaptive overlay, the addition of HyDeA shows consistent improvements, although lower compared to a static gossip overlay. The improvements remain stable for different Playout Delays, and are around 7-8% better than the sole gossip (Figure 7.6 (a)).

However in this scenario the HP peers bandwidth exploitation is greatly reduced (Figure 7.6 (c)) and the chunk loss improvements are more evenly spread across all peer classes (Figure 7.6 (b)). This can be explained by the adaptive mechanism of the gossip overlay who contributes by itself in increasing the upload bandwidth if available.

This is only a first test, we have reasons to believe that a further tuning of the system could further increase, if required, the HP peers upload throughput.

## 7.4 Conclusion

In this chapter we discussed and analyzed the possibility of a hybrid algorithm (HyDeA) for chunk delivery in P2P-TV systems, using either a gossip-based approach or a content-based mechanism for spreading and retrieving chunks. We introduced two technologies to optimize content-based delivery real-time scenarios: pseudo-cache and bandwidth-selective peer join. They are able to (*i*) speed up chunk diffusion and lower the signalling overhead for content-driven retrieval, and (*ii*) increase bandwidth exploitation, thus decreasing chunk losses. The algorithm proposed is easy to implement over existing gossip protocols, and is able to maintain backward compatibility. Simulation results not only show that such a system consistently improves the performances of gossip-based systems, but also demonstrate that integrating a retrieval mechanism based on traditional DHT can not improve gossip delivery in real time contexts. In fact, when the playout delay is particularly small, the content-driven lookup mechanism has to be sped up in order to cut logarithmic DHT routing, allowing for on-time retrieval of requested chunks. Furthermore, having a pull mechanism targeting only high performing peers allows for a better exploitaition of their spare upload bandwidth to the advantage of the least performing ones, who would otherwise be penalized by the tit-for-tat policy, without noticeable consequences on the rest of the overlay. The system has been tested with an extensive simulation campaign, and a real implementation is under development.

Chapter 8

# CCN-TV: a data-centric approach to real-time video services

## Contents

Content-Centric Networking (CCN) is a promising data-centric architecture, based on in-network caching, name-driven routing, and receiver-initiated sessions, which can greatly enhance the way Internet resources are currently used, making support for a broader set of users with increasing traffic demands possible. The CCN vision is, currently, attracting the attention of many researchers across the world, since it has all the potential to become ready to the market, to be gradually deployed in the Internet of today, and to facilitate a graceful transition from a host-centric networking rationale to a more effective data-centric working behaviour. At the same time, several issues have to be investigated before CCN can be safely deployed at the Internet scale. They include routing, congestion control, caching operations, name-space planning, and application design. With reference to application-related facets, it is worth noticing that the demand for TV services is growing at an exponential rate over time, thus requiring a very careful analysis of their performance in CCN architectures. To this end, in the present contribution we deploy a CCN-TV system, capable of delivering real-time streaming TV services, and we evaluate its performance through a simulation campaign based on real-world topologies.

## 8.1 Introduction and Related Work

Due to the relevant importance that content sharing applications are going to play in the upcoming future [Cis 2011, Ahlgren 2011], the Content Centric Networking (CCN) rationale [Jacobson 2009b] has been proposed as a possible way to drive the current *host-centric* Internet paradigm towards a novel *content-centric* behaviour. It is based on in-network caching operations, receiver initiated data exchange, hierarchical content naming, and native support to security and privacy.

In a CCN, contents are split in chunks which are requested using opposite *Interest* messages, generated at the client side. Each *Interest* is then routed until it reaches a node which has, in its own cache, a copy of the requested item. Then, this copy is sent, as a *Data* message, back along the path the *Interest* had gone through. Intermediate nodes can cache the *Data* before forwarding it to the next node (more details on the CCN working behavior is provided in Section 8.2).

Since its birth, the CCN vision has gained a warm attention from both scientific and industrial communities to discover the bounds of its real potential from different perspectives. Many studies have focused on modeling and designing caching strategies and data-transfer performance such as in [Carofiglio 2011a]- [Rossi 2012]. In that direction, it is now clear that the cache size may have a major impact on the overall performance of a CCN even if finding an optimal caching strategy is still an open problem to address. With respect to congestion control issues, instead, recent studies show as the classic additive increase multiplicative decrease algorithm, at the foundation of TCP, could be inherited by CCN, provided that some countermeasures are employed to limit unfairness issues that could arise among contents with different popularities [Grieco 2012, Carofiglio ]. Another very relevant topic in CCN research covers routing operations, which are essential to properly drive the dissemination of receiver generated *Interest* packets. To this end, the adoption of Bloom filters appears a promising solution [Tortelli 2012, You 2012] that merits further investigations. Starting from this premise, it is evident that all facets of CCN are going to be afforded in an ebullient panorama of activities that cover both the underlying mechanisms within the protocol architecture and the design of content oriented applications and services. With reference to application-related features, it is worth to notice that the demand for TV services is growing at an exponential rate over the time [Cis 2011], thus requiring a very careful analysis of their performance in CCN architectures. A preliminary study presented in [Li 2011] addresses time shifted applications only, whereas live streaming operations are currently under investigation as testified in the interesting contribution [Xu 2012]. To complement the research efforts of the community in a so relevant domain, the present manuscript is intended to design a complete CCN-TV system encompassing all the main facets of typical live streaming video services. The proposed CCN-TV has been tested through a solid simulation campaign based on real topologies. To this end, the *ccn-Sim* simulator [Rossini 2012] has been properly tailored to our purposes by adding window based flow control, handling of playout delay and real-time data, advanced logging functions, links with bandwidth constraints, and data session bootstrapping. Simulation results shown that in-network caching seems to play a minor role in live streaming video services, mainly because cached data looses its utility after the deadline is expired. On the other hand, the way CCN handles client requests for TV contents helps improving the performance of the system with respect to a plain IP infrastructure.

## 8.2 Basic background on CCN

Internet usage has undergone a radical change during the last ten years: content-sharing applications are now dominant whereas the IP architecture still provides a connection-less service among remote hosts [Ahlgren 2011]. Users ask for contents, looking for *what* they intend to retrieve from the Internet while the language spoken by the underlying IP infrastructure provides answers on *where* a packet should be sent. This mismatch is actually overcome by a number of workarounds at different levels of the protocol stack, which, indeed, limit the overall efficiency of the Internet.

The so-called *Future Internet* represents a family of possible solutions to the aforementioned issues, embracing novel communication models that can better accommodate and fulfill users' requirements related to efficiency, security, support to mobility, and integrated media experience [Ahlgren 2011].

At the present stage, many valid proposals for the Future Internet exist, such as the Publish Subscribe Internet Routing Paradigm, the 4WARD NetInf project, and the Cache-and-Forward Network Architecture, the Data-Oriented Network Architecture and the CCN approach [Jacobson 2009b, Ahlgren 2012], having different levels of compatibility with the IP paradigm.

Among them, the CCN vision looks promising since, besides being "data-centric", it can be gracefully integrated with today's IP-based Internet. In a CCN, all content is unambiguously identified by a hierarchical name, allowing users to retrieve information without being aware about the physical location of servers (e.g. IP address). Also, commu-nication is receiver-driven and based on content chunk exchange, name-based routing, and self-certifying packets [Jacobson 2009b].

Nevertheless, the real performance bounds of a CCN and the actual benefits it can bring to the Internet are still not entirely known, mainly because there are many open issues that surround the CCN architecture, such as those related to: (i) routing, (ii) congestion control, (iii) strategy layer design, (iv) name space definition, (v) semantic layer, (vi) accurate models, and (v) fairness among heterogeneous applications and contents having different popularities.

As specified before, CCN communications are driven by the consumer of data and only two types of messages are exchanged (namely *Interest* and *Data*). A user may ask for a content by issuing an *Interest*, which is routed within the CCN towards the nodes in posses of the required information, thus triggering them to reply with *Data* packets.

The routing operations are performed by the strategy layer only for *Interest* packets. *Data* messages, instead, just follow the reverse path to the requesting user, allowing every intermediate node to cache the forwarded content.

CCN adopts a hierarchical structure for *names*, which leads to a *name tree*. In particular, it is formed by several components, each one made by a number of arbitrary octets (optionally encrypted), so that every *name prefix* identifies a subtree in the name space. An *Interest* can specify the full name of the content or its prefix, thus accessing to the entire collection of elements under that prefix.

Finally, since contents are exchanged based on their names, multiple nodes interested in a particular data can share it using multicast suppression techniques over a broadcast medium. Analyzing a CCN node, it is possible to identify three main structures [Jacobson 2009b].

- the *Content Store* (CS): a cache memory that can implement different replacement policies as Least Recently Used (LRU) and Least Frequently Used (LFU);
- the *Forwarding Information Base* (FIB): is similar to an IP FIB except for the possibility to have a list of *faces*[1] for each Content Name entry, thus allowing *Interest* packets to be forwarded towards many potential sources of the required Data;
- the *Pending Interest Table* (PIT): is a table used to keep track of the *Interest* packets that have been forwarded upstream towards content sources, combining them with the respective arrival faces, thus allowing the properly delivery of backward Data packets sent in response to *Interests*.

When an *Interest* packet arrives to a CCN node, the CS is searched to discover whether a data item is already available as an answer to be sent immediately back to the requesting user. Otherwise, the PIT is consulted to find out if others *Interest* packets, requiring the same content, have been already forwarded towards potential sources of the required data. In this case, the *Interest*'s arrival face is added to the PIT entry. Otherwise, the FIB is examined to search a matching entry, indicating the list of faces the *Interest* has to be forwarded through. At the end, if there is not any FIB entry, the *Interest* is discarded.

On the other hand, when a Data packet is received, the PIT table comes into play, which, keeping track of all previously forwarded *Interest* packets, allows to establish a backward path to the node that requested the data.

## 8.3 CCN-TV architecture

Unlike Video-On-Demand, real-time video distribution has to deal with a specific class of problems to ensure the timely delivery of an ordered stream of chunks. Video chunks have to be received in playing order and within a given time interval (the

[1] In CCN it is used the term "*face*" instead of the "*interface*" because packets are also exchanged between application process, besides being forwarded only over real network interfaces.

*playout delay*), before they are actually played, thus “expiring”. A chunk not delivered before its expiration will result in degradation of the rendered video, impacting the end user Quality of Experience; the extent of the video degradation may vary depending on the video codec and the type of the lost frame. To solve these challenges, client nodes implement a receiving buffer queue where the chunks are stored in order, that is emptied while the video is being played. Therefore, any chunk not received before its playing instant becomes useless. To reduce the chance of chunk loss several mechanisms can be put in place to retransmit requests for chunks close to expiration. Furthermore, in modern codecs, such as H.264 [Wiegand 2003], there are different types of video frames, encoded using intra-frame or inter-frame techniques, each having a different level of importance. For example, the so called I-frames, derived using intra-frame compression techniques, actually represent a full video image, providing a fundamental reference for subsequent inter-frame encoded images.

With this in mind in CCN-TV we considered a network of nodes requesting different real-time video streams, identified by a *channelID*, served by one or more broadcast server.

Unlike canonical UDP/TCP-based streaming, in CCN-TV each video chunk, identified by a progressive *chunk number*, has to be requested individually, via a dedicated *Interest*.

Although this might look costly at a first sight, CCN’s routing mechanisms ensure that *Interests* for the same chunk do not propagate twice along the same routing path (unless under specific conditions, as explained in Section 8.3.2), and the caching strategy implemented by every node can make sure that *Interests* for the most popular contents are served before going through the whole routing path. Moreover, the *Interest/data* exchange allows for a natural flow control mechanism, where each node can request for new chunks just when the old ones have arrived.

Herein we thoroughly describe the design rationale and all the details of the CCN-TV system this work targets. Specifically, in what follows, we present: the bootstrap phase, the flow control strategy, and the management of retransmitted *Interest* packets.

### 8.3.1 Channel bootstrap

One challenge we are faced with in a real-time scenario is to bootstrap the channel to be received. Bootstrapping a channel involves the operations of finding a routing path to the nearest channel provider and locating the first valid chunkID of the video stream. Due to video codec re-quirements, the video stream can be visualized only once the first I-frame has been received. Therefore, a client has to first gather from the server the first chunk (and the corresponding chunkID) of the last generated I-frame. To do so, it sends an *Interest* packet for the URI: `[domain]/[channelID]`, with the *Status* field set to *BOOTSTRAP* and the *Nonce* field set by the client, as in in Section 8.3.4. An *Interest* with Status = BOOTSTRAP would travel unblocked until it reaches the first good stream repository (i.e. a node who can provide a

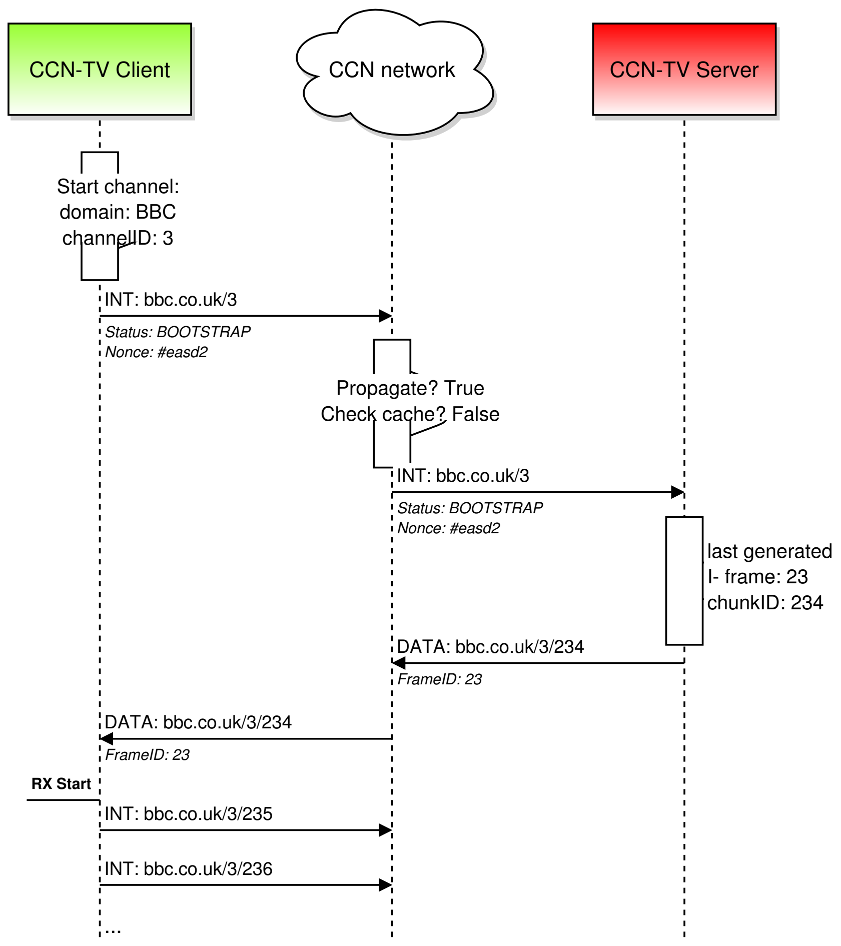


Figure 8.1: Bootstrap handshake

continuous real-time flow of chunks, not just cached ones).

To this *Interest*, the server responds with a data message in the format `[domain]/[channelID]/[chunkID]`, with *chunkID* being the first chunk of the last generated I-frame, and the corresponding *Frame ID* field value. Upon receipt, the node starts asking for subsequent chunks, using the sliding window mechanism detailed in Section 8.3.2. The use of a *nonce* (a uniquely generated identifier) in the *Interest* URI allows the *Interest* to propagate to the server without being blocked along the routing path, as every bootstrap *Interest* for the same channel has a different nonce. It also avoids the retrieval of the data response from the cache of an intermediate node; the risk, in this case, is the retrieval of a bootstrap data message for a given channel from a cache containing an already expired chunk of an I-frame.

### 8.3.2 Flow control

From the moment a node receives the bootstrap data message, it can initiate the sliding window mechanism to request the subsequent chunks in an optimal way. Each node has a windows of size $W$ to store W pending chunk. We define *pending chunk* a chunk whose *Interest* has been sent by the node, and the window containing the pending chunks a *Pending Window*. Together with the chunkID, we store in the pending window other information, such as the timestamp of the first request and the timestamp of the last retransmission. Whenever a new data message is received, the algorithm described in Figure 8.2 runs over the Pending Window, to perform the following operations:

1. Purge the Pending Window from all the chunks who are expired, i.e., who have already been played, to free new space in the sliding window.

2. Retransmit all chunks that have not been received for a given timeout (onward denoted as *windowTimeout*.

3. Transmit, for each slot that got freed by the received or expired chunks, the *Interest* for a new one.

```
procedure SendInterest(PendingWindow, W, WinTimeout, Now, LastChunkID)
  for each ChunkID in PendingWindow do
    if ChunkID is expired                                  Remove all expired pending Interests.
      Remove ChunkID from PendingWindow
    endif
    if ChunkID.lastTransmissionTime < (Now - WindowTimeout)      Resend stale Interests.
      resend(Interest( ChunkID))
       ChunkID.lastTransmissionTime = Now
    endif
  done
  NumberNewChunks = W - size( PendingWindow )              Send new Interests to fill free slots.
  for i in 1 ..NumberNewChunks do
    send(Interest( LastChunkID ))
    LastChunkID.lastTransmissionTime = Now
    PendingWindow.add( LastChunkID )
    LastChunkID++
  done
```

Figure 8.2: Sliding window algorithm

Furthermore, the same operations are performed if a node doesn't receive any data for at least *windowTimeout* seconds, in which case, all the *Interests* for non-expired chunks in the Pending Window are retransmitted, together with new chunks if new slots have been freed due to expired chunks.

To provide a further insight, we reported in Fig. 8.3 an example of the conceived sliding window algorithm, in which we have set the value of $W$ to be equal to 3.

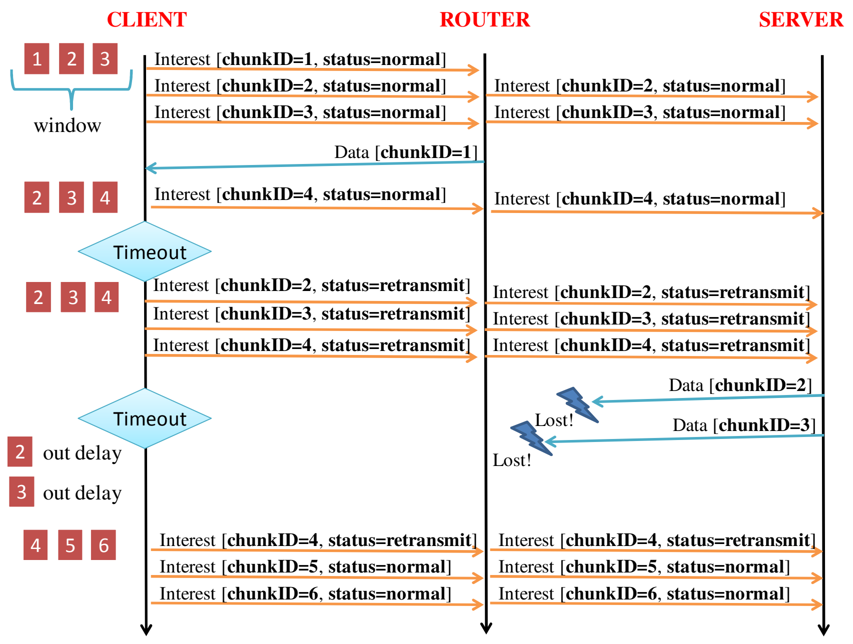

CLIENT
ROUTER
SERVER
1 2 3
window
Interest [chunkID=1, status=normal]
Interest [chunkID=2, status=normal]
Interest [chunkID=3, status=normal]
Interest [chunkID=2, status=normal]
Interest [chunkID=3, status=normal]
Data [chunkID=1]
2 3 4
Interest [chunkID=4, status=normal]
Interest [chunkID=4, status=normal]
Timeout
2 3 4
Interest [chunkID=2, status=retransmit]
Interest [chunkID=3, status=retransmit]
Interest [chunkID=4, status=retransmit]
Interest [chunkID=2, status=retransmit]
Interest [chunkID=3, status=retransmit]
Interest [chunkID=4, status=retransmit]
Data [chunkID=2]
Lost!
Data [chunkID=3]
Lost!
Timeout
2 out delay
3 out delay
4 5 6
Interest [chunkID=4, status=retransmit]
Interest [chunkID=5, status=normal]
Interest [chunkID=6, status=normal]
Interest [chunkID=4, status=retransmit]
Interest [chunkID=5, status=normal]
Interest [chunkID=6, status=normal]

Figure 8.3: Sliding window example

### 8.3.3 Interest routing

As described in Section 8.2, CCN nodes along the routing path of an *Interest* will stop the propagation of said *Interest*, if they have previously routed another *Interest* for the same resource, and the correspondent data has note been sent back yet; instead, they will simply update their Pending Interest Table adding the face from where this newcomer *Interest* was originated, so to reroute the data back recursively along the path the *Interest* has gone through.

This mechanism ensures flow control and limits the propagation of duplicate *Interests*, in case several nodes in the same network are watching the same channel.

However, to make the *Interest* retransmission mechanism effective, a retransmitted *Interest* needs to propagate all the way up to the server, or to the first node with the desired chunk in cache. Therefore, retransmitted *Interests* carry the *Status* field set to *Retransmission* to mark if the *Interest* is a retransmission or not, and each node along the routing path propagates the *Interests* marked as retransmitted, thus skipping the usual CCN mechanism, unless the correspondent chunk is found in the cache.

### 8.3.4 CCN-TV messages

As detailed above, additional functionalities required by the system for real-time video streaming are implemented on top of existing CCN data and *Interest* messages via new fields carrying the required additional information. However, should the situation require the system to conform to classical CCN messages, all additional fields can be easily replaced by additional fields in the content name.

| Packet type | Field | Content |
|---|---|---|
| *Chunk Interest* | *Content Name* | [*domain*]/[*channelID*]/[*chunkID*]. |
| | *nonce* | Used only for the bootstrap phase. |
| | *Publisher Filter* | Not used. |
| | *Status* | *Bootstrap*, *Normal*, *Retransmission*. |
| *Chunk Data* | *Content Name* | [*domain*]/[*channelID*]/[*chunkID*]. |
| | *Publisher ID* | Optional. |
| | *Signature* | Optional (for increased content authentication). |
| | *Stale Time* | Set to the *frame time* of the frame the chunk belongs to. |
| | *Frame ID* | ID of the frame the chunk belongs to. |
| | *Data* | The request video chunk binary data. |

Table 8.1: Messages used in CCN-TV

Table 8.1 shows how we made use of the classical CCN message fields, together with the new fields and their use. In particular, CCN-TV *Interests* carry an additional *Status* fields marking if the *Interest* is a *bootstrap Interest* (Section 8.3.1), a normal one or a *retransmission* (Section 8.3.3). Concerning CCN data message, we extended the messages with an additional field, i.e., *Frame ID*, containing the ID of the frame to which the embedded chunk belongs to.

## 8.4 Simulation results

In this section, we will evaluate performances of the proposed CCN-TV architecture. To this end, we implemented it within *ccnSim*, i.e., an open source and scalable chunk-level simulator of CCN [Rossini 2012] built on top of the Omnet++ framework [omn ], dedicated to the evaluation of Video On Demand systems on top of CCN. By itself, *ccnSim* models a complete video distribution systems, with a high degree of fidelity concerning catalogs, requests and repositories distribution, and network topologies. Since, however, it did not support the real-time constraints required by our evaluations, it has been modified and improved in the following ways:

- we added support for links with bounded capacity and packets with a well defined size, which was missing in *ccnSim*, to be able and estimate the CCN behavior under some bandwidth constraints;
- due to the datarate channels, we implemented a transmission queue for each face of each node, in order to properly manage the packet transmission;
- we added the support for synthetic video traces, so to be able to transmit and receive chunk of real videos, and consequently being able to reconstruct the received video and evaluate its PSNR;

- due to possible expiration of *Interests*, we implemented a cleanup mechanism for each node's PIT, to avoid having in long term stale entries due to expired chunks;
- we improved and enriched the logging system, so to be able to record each node's received chunks and reconstruct the received video;
- we added more controls server-side, to send a data only for those chunks who have already been generated.

Furthermore, the following mechanisms, beyond the provided ones, have been implemented in the simulator:

- the sliding window mechanism described above, and all the related data structures;
- the *Interest* forceful propagation in case of retransmission;
- constant data reception, until a channel is changed.

The extended *ccnSim* simualtor is available at [ccn ].

The aim of our study is to evaluate how the behavior of the CCN-TV system is influenced by (i) the amount of the network bandwidth dedicated to real-time streaming services, (ii) the *windowTimeout* adopted by the sliding window mechanism, (iii) the playout delay, and (iv) the cache decision policy.

We focus the attention on the GEANT network, which interconnects the European research and education institutions and it is composed by 22 routers [GEA ]. Every node of the network is considered to be a direct CCN node, i.e. no TCP or UDP encapsulation is implemented. We assume the presence of only one small video streaming provider that offers 5 parallel real-time transmissions to remote clients. It is connected to one of the nodes forming the GEANT topology. In every simulation round, each video content is mapped to a video stream compressed using H.264 [Wiegand 2003] at a average coding rate randomly chosen in the range [250, 2000] kbps. Clients, i.e., CCN nodes that download video contents from the server, are connected to remaining nodes (1 client per node). In order to catch the behavior of people watching TV, we modeled two groups of users: *faithful* and *zapping*. *Faithful* users are attached to one video channel for the whole simulation. *Zapping* users, instead, change frequently the channel among those offered by the server according to a Poisson process with parameter $\lambda = 0.0666$. Further, the channel selection process has been modeled considering that contents popularities follow a Zipf distribution. According to [Rossi 2011], the most of works presented in literature set the parameter $\alpha$ of the Zipf distribution in the range [0.6, 2.5]. In line with these common settings, we set $\alpha = 1$. Once a client decides to watch a specific channel, it performs the bootstrap process described in the previous section and then starts sending *Interest* packets following the designed sliding window mechanism.

In our tests, we adopted the optimal routing strategy, already available within the *ccnSim* framework. According to it, *Interest* packets are routed towards the video server along the shortest path. On the other hand, three caching strategies have been considered in our study: *no-cache*, *LRU*, and *FIFO* [Rossi 2011]. When well known *LRU* or *FIFO* policies are adopted, we set the size of the cache to 100 chunks. The *no-cache* policy is intended to evaluate the performance of the CCN without using any caching mechanism.

The window size $W$ has been set to 10, ensuring that faces of the server are almost fully loaded in all considered scenarios. Also, the transmission queue length associated to each face, $Q$, has been set, in order to be larger than

$$Q = 2 * L_c * P_D \tag{8.1}$$

where $L_c$ and $P_D$ represent link capacity and maximum propagation delay in the considered network topology. All simulation parameters have been summarized in Table 8.2.

### 8.4.1 Interest generation process

As a first step, we investigate the impact that the sliding window mechanism has on the amount of sent *Interest* packets, which is shown in Figure 8.4. From these plots it is evident that the highest *windowTimeout*, the lowest the total number of *Interest* messages sent by end users. When the *windowTimeout* increases, the probability that a given client does not receive any chunks within such a time interval decreases and, as a consequence, also the number of retransmitted *Interest* packets decreases as well. As a further confirm of this result, Figure 8.5 shows that the percentage of duplicated *Interest* packets increases when the *windowTimeout* decreases due to a high number of chunks that are unable to reach the client within the expected timeliness.

As expected, the *playout delay* has a minor impact on the number of generated *Interest* messages, which, as is known from the theory on sliding window mechanisms [Kurose 2012], can be only influenced by window size ($W$) and *windowTimeout*.

Also, caching operations do not have any significant impact on the number of generated *Interest* messages. The main reason being that chucks stored in cache memories lose their effectiveness after their deadline is expired.

On the other hand, the link capacity greatly influence the *Interest* transmission rate. From Figure 8.4 emerges, in fact, that the number of *Interest* lowers when the capacity of links decreases. This is because a limited bandwidth reduces the quota of received chunks, thus preventing a rapid advancement of the sliding window. In other terms, this result proves, once again, the effectiveness of the sliding window mechanism in CCN.

### 8.4.2 QoS and QoE

The first important parameter that describes how CCN-TV settings affect the quality of service offered to end users is the chunk loss ratio, which represents the

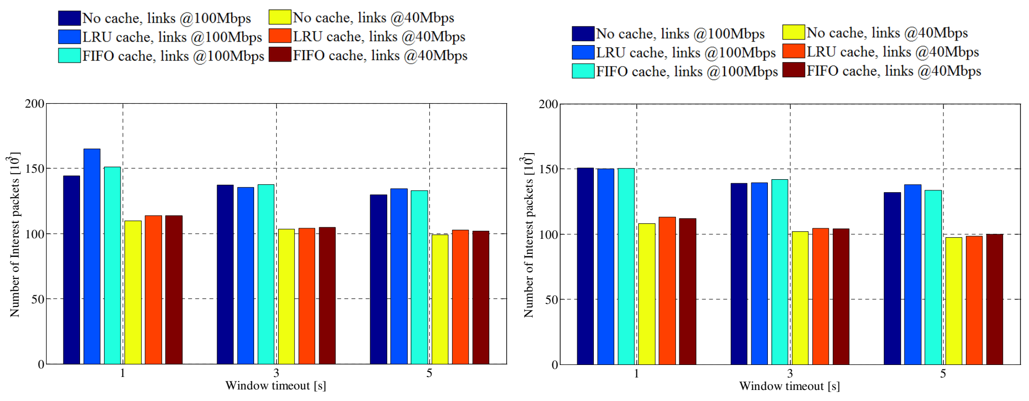
Table 8.2: Summary of simulation parameters

| **Parameter** | **Value** |
|---|---|
| Topology | GEANT with 22 routers |
| Link capacity | 40 Mbps and 100 Mbps |
| Number of real-time service provides | 1 |
| Number of clients | 21 |
| Catalog size | 5 files |
| Chunk size | 10Kbytes |
| Video average bit rate | 250kbps, 600kbps, 1000kbps and 2000kbps |
| W (window size) | 10 |
| Playout delay | 10s and 15s |
| Window timeout | 1s, 3s, and 5s |
| Caching strategy | No cache, LRU, and FIFO |
| Cache size | 100 chunks |
| Client zapping behavior | 50% fixed, 50% changing on average every 15s |
| Simulation time | 60s |
| Number of seeds | 5 |

percentage of chunks that have not been received in time (i.e., before the expiration of the *playout delay*) by clients.

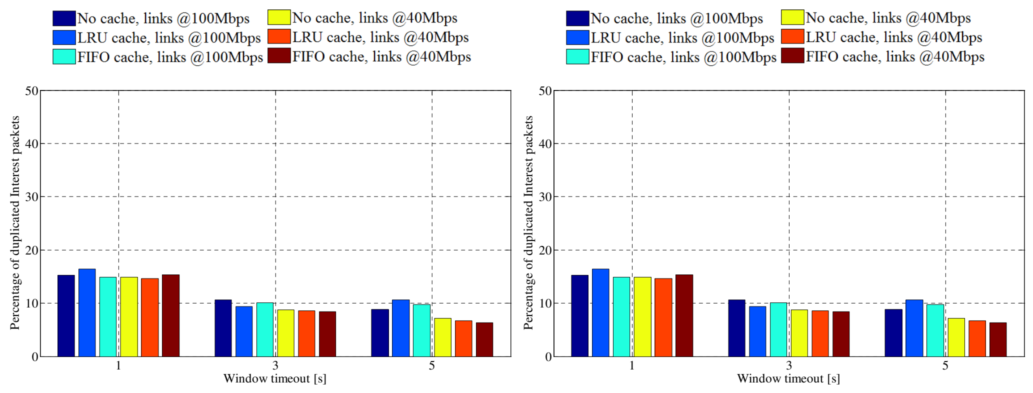
From Figure 8.6, showing the chunk loss ratio measured in all considered network scenarios, we note that *playout delay* plays a fundamental role. When the *playout delay* increases, in fact, the client could receive a *Data* packet within a longer time interval, thus reducing the amount of chunks discarded because out of delay. On the other hand, a slight increment of the chunk loss ratio can be registered by increasing the *windowTimeout*. If the client retransmits an *Interest* packet after long time, there is the risk that the *Data* packet will be reached by the destination after the expiration of the *playout delay*. In addition, we note that a reduction of the link capacities leads to a higher number of lost chunks, due to increased latencies induced by network congestion.

It is very important to remark that the presence of the cache can guarantee only a small reduction of the chunk loss ratio. With our study, we found that, in the presence of real-time flows, the cache does not represent an important CCN feature. On the other hand, we noticed that the PIT plays a more relevant role. In fact, in presence of live video streaming services, clients that are connected to a channel request the same chunks simultaneously. In this case, a CCN router has to handle multiple *Interest* messages that, even though sent by different users, are related to the same content. According to the CCN paradigm, such a node will store all of these requests into the PIT, waiting for the corresponding *Data* packet. As soon as the packet is received, the router will forward it to all users that have requested the chunk in the past. According to these considerations, the use of the cache will not



(a) (b)

Figure 8.4: Total number of *Interest* packets sent by clients with playout delay of (a) 10s and (b) 15s.



(a) (b)

Figure 8.5: Percentage of duplicated *Interest* packets sent by clients with playout delay of (a) 10s and (b) 15s.

produce a relevant gain of network performances. Indeed, the PIT helps reducing the burden at the server side by avoiding that many *Interest* packets for the same chunk are routed to the server.

To conclude our study, we have computed Peak Signal to Noise Ratio (PSNR), which is nowadays one of the most diffused metrics for evaluating user satisfaction, together with interactivity level, in real time video applications [Piro 2011]. Results shown in Figure 8.7 are in line with those reported for chunk loss ratio (the PNSR is higher in the same case in which the chunk loss ratio is lower). Again, link capacity greatly influences the quality of the TV service provided to users. According to [Ohm 2004], the obtained PSNR values can be translated to a Mean Opinion Score (MOS) not less than 4, corresponding to satisfactory quality for almost all users.

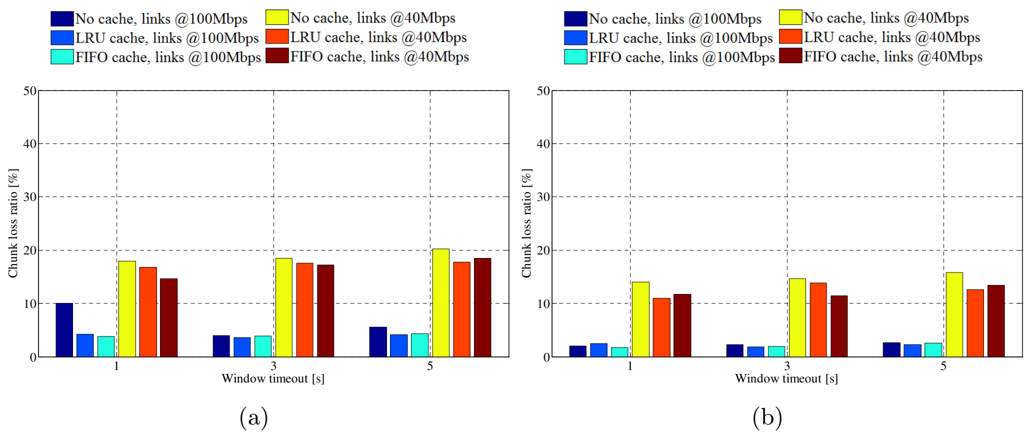


Figure 8.6: Chunk loss ratio with playout delay of (a) 10s and (b) 15s.

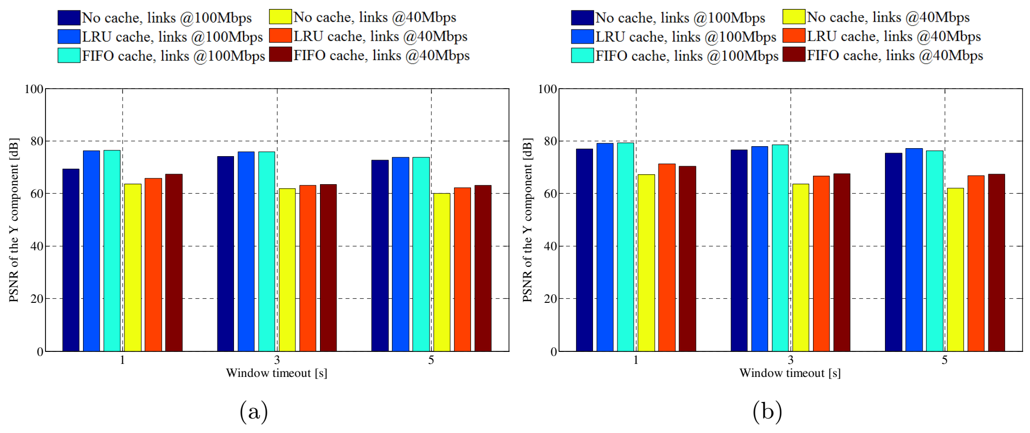


Figure 8.7: PSNR of the Y components of received videos with playout delay of (a) 10s and (b) 15s.

## 8.5 Conclusion

In this chapter, the effectiveness of TV services in a CCN has been investigated. To this end, the *ccnSim* simulator has been modified to add several relevant features such as window-based flow control, handling of playout delay and real-time data, advanced logging mechanisms, and data session bootstrapping. Preliminary results reported herein clearly show that the most relevant CCN feature to TV services is the management of *Interest* packets through the PIT data structure. In fact, such a mechanism limits the number of requests for the same chunk at the server side for multiple clients watching the same TV channel, thus decreasing the link and the computational load at the server.

The contributions of this chapter have been published as *CCN-TV: a data-centric approach to real-time video services*, in Proceedings of Advanced Information Networking and Applications (AINA), 2013, IEEE [Ciancaglini 2013].

# Part IV

# Conclusion

Chapter 9

# Summary and concluding remarks

In this thesis, we have attempted to tackle different kinds of problems related to content-addressable routing systems, with focus on key-based routing systems, in the form of Structured Overlay Networks and the Future Internet architecture of Content-Centric Networks.

In particular, we have covered two different axes, related to the co-operation of heterogeneous structured overlay networks, and the exploitation of key- and content-based routing in the real-time distribution of video content.

**Overlay cooperation.** The first covered topic deals with enabling co-operation for heterogeneous structured overlay systems. The idea stems from several observations, the first one being the presence of several competing overlay architectures both in research and on the market, each of these architectures having its own strengths and weaknesses, and some heavily specialized for a particular class of applications or data structures. The second observation is that many of these systems need to deal with severe performance degradation in real-world scenarios that impair the overall performance due to network artifacts, an impairment often proportional to the size of the network. With this in mind, it is clear that the ability of interconnecting several overlay networks efficiently and transparently becomes desirable, in the perspective of both being able to design a new class of distributed applications supporting, for example, different types of data or different classes of nodes (desktop, server, mobile, ...), and being able to organize a large-scale network around smaller clusters, centered around local properties of the data they manage. With regard to these challenges, we presented a novel architecture that allows for transparent interconnection of heterogeneous structured overlays, by exploiting co-located nodes as a form of distributed gateways. Nodes in the network can efficiently discover gateways to other networks by analyzing the underlying overlay traffic, or via active notifications, in those cases where backward compatibility with existing peers in the network has to be maintained. Together with a detailed description of the architecture and an extensive evaluation by means of simulation and real-world deployment, we provided a first mathematical model to aid the design and performance evaluation of inter-connected systems of extremely large scale, conditions where the aforementioned evaluation methods would be unfeasible. Furthermore, we provided two examples of applications relying on a network of interconnected overlays, in the form of proofs-of-concept, to show the potential and opportunities linked to such an approach.

**Content-based techniques in real-time video streaming.** The second topic unravels into two different aspects. First off, we analyzed if and how a key-based routing systems would impact the performances of a mesh-based P2P-TV system. P2P-TV systems have to retrieve content in an orderly and timely manner, while relying on a fixed neighborhood of nodes with which to exchange data. One feature of key-based routing systems, when using consistent hash functions is the generation of pseudo-random traffic that evenly loads all of the peers in a system. With this in mind, we tried to apply this concept to a P2P-TV system, in order to provide each peer in the system with a "virtual neighbor", reachable by issuing content-addressable requests for the content close to expiration that have not been retrieved yet by the mesh neighborhood. We analyzed the system under stressful conditions and network load, implementing our content-based algorithm on top of the NAPA-Wine network simulator, and compared it with both a static and an adaptive mesh topology, to understand if such a solution would provide enough benefits to the overall system. We then moved on to a full content-centric approach, by implementing and evaluating a complete real-time video streaming systems on top of the novel Content-Centric Networks architecture. The idea was to determine what would be the challenges of implementing such system on top of a complete host-less architecture such as CCN, adapting video streaming to CCN protocol packets, and to analyze how the built-in caching mechanism would affect a distribution of real-time, expiring content. The system, named CCN-TV, has been implemented on top of ccn-sim, a CCN network simulator developed on top of the Omnet++ framework, and extensive analysis in terms of overall traffic and Quality of Experience has been provided.

## 9.1 Future directions

The following are some future research directions concerning the work presented in this thesis:

**Extension to unstructured overlays.** The extension to interconnect unstructured overlay and devise a form of cooperation between structured and unstructured overlays seems like a natural direction for the present research work. However, while interconnecting different systems all based on the key-based routing approach provides a common ground amongst all the networks, unstructured overlays usually rely on a keyword-based search paradigm, which would require a more careful design, involving all the system tiers up to the application layer, to be effectively integrated with key-based routing systems.

**Overlay self-organization.** The work done so far in Synapse involved mainly defining and describing extensively the meta-protocol and proving the feasibility of interconnected systems. As of now, there are still several interesting research issues open: for example, how to organize a set of overlays around the data, and how to maintain self-organizing properties in the systems. With regard to this, Rodriguez et al. [Pujol ] proposed an algorithm for social graph partitioning that aims at minimizing, rather than the number of edges between clusters, the number of nodes to be duplicated between clusters in order to reduce redundancy between clusters. While their work was aiming at improving the performance of Facebook's NoSQL database system Cassandra [Lakshman 2010], it could be applied with the intention of achieving self-organization in a peer-to-peer social network that would rely on interconnected overlays to share the data securely just amongst a circle of trusted contacts, while still being able to perform requests throughout the entire system. Another aspect to be improved is the maintenance of the gateway tables: in a high-churn environment, which one can expect in an overlay network, it becomes important to reduce the presence of stale entries in the routing tables of a node and to constantly find the best path to a foreign overlay. To achieve this, a study is undergoing on how to use Cognitive Packet Networks [Gelenbe 1999] in order to maintain an updated list of the best paths for a node, in terms of latency, load, energy, reliability etc., to reach another overlay.

**Better overlay modeling.** The mathematical modeling proposed in this thesis is a first attempt at providing a reliable way to estimate performance parameters for networks with a number of nodes hypothetically going to infinity. In order to achieve this, we need to make several simplifications on the scenario and gateway configuration, simplifications that are currently being under study in a new version of the model. Furthermore, the interest in modeling the interconnection of systems lies beyond the simple overlay network scenario [Gao 2011], and a successful modeling could be applied to fields other than distributed systems.

**P2P-TV improvements.** The analysis work carried on in Chapter 7 showed that a key-based routing support for P2P-TV can provide performance improvements, but only under certain conditions. In case of a uniform set of desktop class machines with DSL connections, the best solution still seems to be an adaptive mesh based network. There are scenarios however where a content-based approach could be successfully exploited, for example in presence of nodes with highly asymmetric bandwidth (i.e. mobile terminals), where this approach could help "circumventing" the tit-for-tat mechanism typical of BitTorrent-like distribution systems to provide the least performing nodes with a CDN-like mechanism to retrieve missing data without impacting the rest of the network. To achieve this, however, there are several steps that need to be undergone: the system need a bandwidth estimation mechanism and a proper incentive mechanism, to build the HP-DHT with the appropriate class of nodes, and further analysis are required to better evaluate the working parameters of such a system.

**CCN-TV improvements.** The study of media streaming over Content-Centric Networks is still a relatively young topic. Further research in the field is being carried out in order to better evaluate the performances and drawbacks of such systems in more complex scenarios, involving a higher number of nodes and content from multiple providers. Another currently ongoing research track involves the analysis of CCN-TV in a crowd-sourced scenario, where several users can choose to watch an arbitrary set of video feeds provided by a crowd attending a common event. Furthermore, the CCN-TV caching layer would need further analysis: at the moment, we are carrying out a research in order to design an optimized priority cache that would take into account the type of frame being cached (I-, B, or P-frame) and consequently manage its expiration and its priority in the transmission queue.

# Bibliography

[a] *Equipage 06 website.* http://www.equipage06.fr/. (Cited on page 80.)

[b] *Google Maps website.* http://maps.google.com. (Cited on pages 82 and 84.)

[c] *NFC Forum website.* http://www.nfc-forum.org/. (Cited on page 81.)

[d] *Otto et co. website.* http://www.ottoetco.org/. (Cited on page 80.)

[Aberer 2003] 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., vol. 32, no. 3, pages 29–33, 2003. (Cited on page 90.)

[Aberer 2005] K. Aberer, L.O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi and M. Hauswirth. *The essence of P2P: a reference architecture for overlay networks.* In Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on, 2005. (Cited on pages vii, 2, 3 and 7.)

[Ahlgren 2011] B. Ahlgren, P. A. Aranda, P. Chemouil, S. Oueslati, L. M. Correia, H. Karl, M. Sollner and A. Welin. *Content, connectivity, and cloud: ingredients for the network of the future.* IEEE Commun. Mag., vol. 49, no. 7, Jul. 2011. (Cited on pages 122 and 123.)

[Ahlgren 2012] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher and B. Ohlman. *A survey of information-centric networking.* Communications Magazine, 2012. (Cited on page 123.)

[Ali 2006] S. Ali, A. Mathur and H. Zhang. *Measurement of Commercial Peer-To-Peer Live Video Streaming.* In Proc. of ICST Workshop on Recent Advances in Peer-to-Peer Streaming, 2006. (Cited on page 104.)

[Baumgart ] I. Baumgart, B. Heep and S. Krause. *OverSim: A Flexible Overlay Network Simulation Framework.* In Proc. of IEEE GI '07. (Cited on pages 9 and 52.)

[Bertinat 2009] M. E. Bertinat, D. De Vera, D. Padula, F. Robledo, P. Rodriguez-Bocca, P. Romero and G. Rubino. *GoalBit: The First Free and Open Source Peer-to-Peer Streaming Network.* In Proc. of 5th international IFIP/ACM Latin American conference on Networking, 2009. (Cited on page 105.)

[bit ] *Bittorrent website.* http://www.bittorrent.com. (Cited on page 2.)

[Bolla 2009] R. Bolla, R. Gaeta, A. Magnetto, M. Sciuto and M. Sereno. *A measurement study supporting P2P file-sharing community models.* Computer Networks, vol. 53, no. 4, pages 485–500, 2009. (Cited on page 68.)

[Brampton 2006] A. Brampton, A. MacQuire, I. A. Rai, N. J. P. Race and L. Mathy. *Stealth distributed hash table: a robust and flexible super-peered DHT*. In Proceedings of the 2006 ACM CoNEXT conference, 2006. (Cited on page 108.)

[Caesar 2006] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan and I. Stoica. *ROFL: Routing on Flat Labels*. SIGCOMM Comput. Commun. Rev., vol. 36, no. 4, pages 363–374, 2006. (Cited on page 19.)

[Cappello *et al.* 2005] F. Cappello *et al.* *Grid'5000: a Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform*. In SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing Grid'2005, pages 99–106. IEEE/ACM, 2005. (Cited on pages 9, 10 and 35.)

[Carofiglio ] G. Carofiglio, M. Gallo and L. Muscariello. *Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks*. In Proceedings of ICN Workshop, co-located with ACM SIGCOMM. (Cited on page 122.)

[Carofiglio 2011a] G. Carofiglio, M. Gallo, L. Muscariello and D. Perino. *Modeling data transfer in content-centric networking*. In Int. Teletraffic Congress, (ITC), 2011. (Cited on page 122.)

[Carofiglio 2011b] G. Carofiglio, V. Gehlen and D. Perino. *Experimental Evaluation of Memory Management in Content-Centric Networking*. In IEEE ICC, 2011. (Cited on page 122.)

[Castro 2003] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron and A. Singh. *SplitStream: High-bandwidth content distribution in cooperative environments*. In Proc. of International workshop on Peer-To-Peer Systems, 2003. (Cited on page 105.)

[ccn ] *CCN-TV Webpage*. `http://telematics.poliba.it/ccn-tv/`. (Cited on page 130.)

[Ceballos 2006] M. R. Ceballos and J. L. Gorricho. *P2P file sharing analysis for a better performance*. In Proc. of the 28th International Conference on Software Engineering. ACM, 2006. (Cited on page 104.)

[Chawathe 2005] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker and J. Hellerstein. *A case study in building layered DHT applications*. In SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications, pages 97–108. ACM, 2005. (Cited on page 90.)

[Cheng 2006] 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*. LNCS, vol. 4269, page 258, 2006. (Cited on pages 16 and 18.)

[Cheng 2007] L. Cheng. *Bridging Distributed Hash Tables in Wireless Ad-Hoc Networks.* In Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE, pages 5159–5163, 2007. (Cited on page 17.)

[Ciancaglini 2010] V. Ciancaglini, L. Liquori and L. Vanni. *CarPal: Interconnecting Overlay Networks for a Community-Driven Shared Mobility.* In Proceedings of Trustworthy Global Computing 2010, 2010. (Cited on pages 10 and 90.)

[Ciancaglini 2012a] V. Ciancaglini, R. Gaeta, L. Liquori and R. Loti. *Modeling and analysis of large scale interconnected unstructured P2P networks.* In Proc. of ICPADS '12, 2012. (Cited on pages 10 and 76.)

[Ciancaglini 2012b] V. Ciancaglini, G. N. Hoang and L. Liquori. *Towards a Common Architecture to Interconnect Heterogeneous Overlay Networks.* In Proceedingds of ICPADS conference 2012, 2012. (Cited on pages 9 and 59.)

[Ciancaglini 2012c] V. Ciancaglini, G. N. Hoang, P. Maksimovic and L. Liquori. *An Extension and Cooperation Mechanism for Heterogeneous Overlay Networks.* In Proceedings of HetNETS Workshop, NETWORKING Conference 2012, 2012. (Cited on pages 9 and 59.)

[Ciancaglini 2013] V. Ciancaglini, G. Piro, R. Loti, L. A. Grieco and L. Liquori. *CCN-TV: a data-centric approach to real-time video services.* In Proceedings of Advanced Information Networking and Applications (AINA). IEEE, 2013. (Cited on page 134.)

[Cis 2011] *Cisco Visual Networking Index: Forecast and Methodology, 2010-2015.* White Paper, Jun. 2011. (Cited on page 122.)

[Ciullo 2010] D. Ciullo, M. A. Garcia, A. Horvath, E. Leonardi, M. Mellia, D. Rossi, M. Telek and P. Veglia. *Network Awareness of P2P Live Streaming Applications: a Measurement Study.* IEEE Transactions on Multimedia, no. 12, 2010. (Cited on pages 104 and 117.)

[Crainiceanu 2007] A. Crainiceanu, P. Linga, A. Machanavajjhala, J. Gehrke and J. Shanmugasundaram. *P-ring: an efficient and robust P2P range index structure.* In SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 223–234. ACM, 2007. (Cited on page 90.)

[Crespo 2005] Arturo Crespo and Hector Garcia-Molina. *Semantic Overlay Networks for P2P Systems.* In Agents and Peer-to-Peer Computing. Springer Berlin Heidelberg, 2005. (Cited on page 5.)

[Cudré-Mauroux 2007] P. Cudré-Mauroux, S. Agarwal and K. Aberer. *GridVine: An Infrastructure for Peer Information Management.* IEEE Internet Computing, vol. 11, no. 5, 2007. (Cited on page 5.)

[Cutillo 2009] L.A. Cutillo, R. Molva and T. Strufe. *Safebook: A privacy-preserving online social network leveraging on real-life trust.* Communications Magazine, IEEE, 2009. (Cited on page 5.)

[Dabek ] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz and Ion Stoica. *Towards a Common API for Structured Peer-to-Peer Overlays.* In Proc. of IPTPS '03. (Cited on pages 7, 42 and 53.)

[Dabek 2001] Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris and Ion Stoica. *Wide-area cooperative storage with CFS.* ACM SIGOPS Operating Systems Review, 2001. (Cited on page 5.)

[Datta 2006] A. Datta and K. Aberer. *The challenges of merging two similar structured overlays: A tale of two networks.* In Proc. of IWSOS, 2006. (Cited on page 16.)

[DC ] *The Dublin Core Metadata Initiative.* `http://dublincore.org/`. (Cited on page 92.)

[DeCandia 2007] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels. *Dynamo: amazon's highly available key-value store.* In ACM Symposium on Operating Systems Principles: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, 2007. (Cited on page 4.)

[EAD ] *Encoded Archival Description.* `http://www.loc.gov/ead/`. (Cited on page 92.)

[Erice 2003a] L. G. Erice, E. W. Biersack, K. W. Ross, P. A. Felber and G. U. Keller. *Hierarchical P2P Systems.* In Proc. of Euro-Par, 2003. (Cited on pages 7 and 16.)

[Erice 2003b] L. G. Erice, K. W. Ross, E. W. Biersack, Pascal A. Felber and G. U. K. *Topology-Centric Look-Up Service.* In Proc. of NGC, 2003. (Cited on pages 7 and 16.)

[erl ] *Erlang programming language website.* `http://www.erlang.org/`. (Cited on pages 10 and 68.)

[eur ] *Europeana.* `http://europeana.eu/portal/`. (Cited on page 92.)

[Feldmann 2007] A. Feldmann. *Internet Clean-Slate Design: What and Why?* SIGCOMM Comput. Commun. Rev., vol. 37, no. 3, pages 59–64, 2007. (Cited on page 16.)

[Fortuna 2010] R. Fortuna, E. Leonardi, M. Mellia, M. Meo and S. Traverso. *QoE in Pull Based P2P-TV Systems: Overlay Topology Design Tradeoffs.* In Proc.

of 10th International Conference on Peer-to-Peer Computing, 2010. (Cited on pages 109 and 111.)

[Francis 2000] P. Francis. *Yoid: Extending the Internet Multicast Architecture*. Rapport technique, AT&T Center for Internet Research at ICSI (ACIRI), 2000. (Cited on page 84.)

[frb 1998] *Functional Requirements for Bibliographic Records*, 1998. `www.ifla.org/VII/s13/frbr/frbr.pdf`. (Cited on page 92.)

[Furtado 2007] P. Furtado. *Multiple Dynamic Overlay Communities and Inter-space Routing*. Lecture Notes in Computer Science, vol. 4125, page 38, 2007. (Cited on page 17.)

[Gaeta 2011] R. Gaeta and M. Sereno. *Generalized Probabilistic Flooding in Unstructured Peer-to-Peer Networks*. IEEE Transactions on Parallel and Distributed Systems, vol. 22, pages 2055–2062, 2011. (Cited on page 68.)

[Gaeta 2013] R. Gaeta, V. Ciancaglini, R. Loti and L. Liquori. *Interconnection of large scale unstructured P2P networks: modeling and analysis*. In Proceedings of ASMTA Conference 2013, 2013. (Cited on pages 10 and 76.)

[Ganesan ] P. Ganesan, P. Krishna Gummadi and H. Garcia-Molina. *Canon in G Major: Designing DHTs with Hierarchical Structure*. In Proc. of ICDCS '04. (Cited on pages 7 and 16.)

[Gao 2011] Jianxi Gao, Sergey V Buldyrev, H Eugene Stanley and Shlomo Havlin. *Networks formed from interdependent networks*. Nature Physics, 2011. (Cited on page 139.)

[Garces-Erice 2003] L. Garces-Erice, K. W Ross, E. W Biersack, P. A Felber and G. Urvoy-Keller. *Topology-Centric Look-Up Service*. In NGC'03, Munich, Germany, 2003. (Cited on page 20.)

[Garces-Erice 2004] L. Garces-Erice, P. A Felber, K. W Ross and G. Urvoy-Keller. *Data Indexing in Peer-to-Peer DHT Networks*. In ICDCS 2004, Tokyo, Japan, 2004. (Cited on page 40.)

[GEA ] *Geant project website*. `http://www.geant.net/`. (Cited on page 130.)

[Gelenbe 1999] Erol Gelenbe, Zhiguang Xu and Esin Seref. *Cognitive packet networks*. In Tools with Artificial Intelligence, 1999. Proceedings. 11th IEEE International Conference on. IEEE, 1999. (Cited on page 139.)

[gnu ] *Gnunet website*. `http://www.gnunet.org`. (Cited on page 2.)

[Grieco 2012] L. A. Grieco, D. Saucez and C. Barakat. *AIMD and CCN: past and novel acronyms working together in the Future Internet*. In Proceedings of Capacity Sharing Workshop 2012 (CSWS'12) co-located with ACM SIGCOMM CoNEXT 2012., Dec. 2012. (Cited on page 122.)

[Harvey 2003] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer and A. Wolman. *SkipNet: a scalable overlay network with practical locality properties.* In USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems, pages 9–9. USENIX Association, 2003. (Cited on page 90.)

[i2p ] *Invisible Internet Project Page.* (Cited on page 5.)

[int ] *Internet Stats.* `http://internetworldstats.com`. (Cited on page 111.)

[Jacobson 2009a] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs and R. L. Braynard. *Networking named content.* In Proceedings of the 5th international conference on Emerging networking experiments and technologies, 2009. (Cited on page 105.)

[Jacobson 2009b] Van Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs and R. L. Braynard. *Networking named content.* In Proc. of Conf. Next (CONEXT), Dec. 2009. (Cited on pages 5, 10, 122, 123 and 124.)

[Jeonghun 2008] N. Jeonghun and D. Sachin. *Pseudo-DHT: Distributed Search Algorithm for P2P Video Streaming.* In Proc. of IEEE International Symposium on Multimedia. IEEE Computer Society, 2008. (Cited on page 105.)

[Jimenez a] R. Jimenez, F. Osmani and B. Knutsson. *Connectivity properties of Mainline BitTorrent DHT nodes.* In Proc. of IEEE P2P '09. (Cited on pages 5, 6 and 7.)

[Jimenez b] R. Jimenez, F. Osmani and B. Knutsson. *Sub-Second Lookups on a Large-Scale Kademlia-Based Overlay.* In Proc. of IEEE P2P '11. (Cited on page 7.)

[Junjiro 2006] 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), vol. E89-B, no. 9, pages 2319–2326, 2006. (Cited on page 18.)

[Kurose 2012] James F. Kurose and Keith W. Ross. Computer networking: A top-down approach. Addison-Wesley Publishing Company, 6th édition, 2012. (Cited on page 131.)

[Kwon 2005] M. Kwon and S. Fahmy. *Synergy: an Overlay Internetworking Architecture.* In Proc. of International Conference on Computer Communications and Networks, pages 401–406, 2005. (Cited on page 18.)

[Labovitz 2000] C. Labovitz, A. Ahuja, A. Bose and F. Jahanian. *Delayed Internet routing convergence.* SIGCOMM Comput. Commun. Rev., vol. 30, no. 4, pages 175–187, 2000. (Cited on page 19.)

[Lakshman 2010] Avinash Lakshman and Prashant Malik. *Cassandra: a decentralized structured storage system.* ACM SIGOPS Operating Systems Review, 2010. (Cited on pages 4 and 139.)

[Leonardi 2008] E. Leonardi, M. Mellia, M. Meo and A. P. Couto da Silva. *Bandwidth-Aware Scheduling Strategy for P2P-TV Systems.* In Proc. of 8th International Conference on Peer-to-Peer Computing, 2008. (Cited on pages 104 and 111.)

[Li 2006] J. Li. *Peer-to-Peer multimedia applications.* In Proc. of the 14th annual ACM international conference on Multimedia, 2006. (Cited on page 104.)

[Li 2011] Z. Li and G. Simon. *Time-Shifted TV in Content Centric Networks:the Case for Cooperative In-Network Caching.* In Proc. of IEEE ICC, Jun. 2011. (Cited on page 122.)

[Liquori ] L. Liquori, C. Tedeschi, L. Vanni, F. Bongiovanni, V. Ciancaglini and B. Marinkovic. *Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks.* In Proc. of Networking '10. (Cited on page 9.)

[Liquori 2009] L. Liquori, C. Tedeschi and F. Bongiovanni. *BabelChord: a Social Tower of DHT-Based Overlay Networks.* In 14th Symposium on Computers and Communications (ISCC 2009). IEEE, 2009. Short paper. (Cited on page 18.)

[Liquori 2010] L. Liquori, C. Tedeschi, L. Vanni, F. Bongiovanni, V. Ciancaglini and B. Marinković. *Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks.* In NETWORKING '10: Proceedings of the 9th International IFIP-TC 6 Networking Conference (to appear). Springer-Verlag, 2010. (Cited on page 37.)

[Liu 2008] J. Liu, S. G. Rao, B. Li and H. Zhang. *Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast.* In Proc. of IEEE, Special Issue on Recent Advances in Distributed Multimedia Communications, 2008. (Cited on page 104.)

[Liu 2009] B. Liu, Y. Cui, Y. Lu and Y. Xue. *Locality-Awareness in BitTorrent-Like P2P Applications.* IEEE Transactions on Multimedia, vol. 3, no. 11, 2009. (Cited on page 104.)

[Locher 2007] T. Locher, R. Meier, S. Schmid and R. Wattenhofer. *Push-to-Pull Peer-to-Peer Live Streaming.* In 21st International Symposium on Distributed Computing (DISC), 2007. (Cited on page 106.)

[log ] *HyDeA Simulator Repository.* http://www-sop.inria.fr/lognet/hydea/. (Cited on page 111.)

[Luo 2008] X. Luo, Z. Qin, J. Han and H. Chen. *"DHT-assisted probabilistic exhaustive search in unstructured P2P networks"*. In "Proc. of International Symposium on Parallel and Distributed Processing, 2008.", 2008. (Cited on page 106.)

[Magharei 2007] N. Magharei, R. Rejaie and G. Yang. *Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches*. In IEEE INFOCOM 2007., may 2007. (Cited on page 110.)

[MAR ] *MARC Standards*. `http://www.loc.gov/marc/`. (Cited on page 92.)

[Marinković 2011] B. Marinković, L. Liquori, V. Ciancaglini and Z. Ognjanović. *A Distributed Catalog for Digitized Cultural Heritage*. In Proceedings of ICT Innovationsm 2010, 2011. (Cited on pages 10 and 99.)

[Maymounkov 2002a] P. Maymounkov and D. Mazieres. *Kademlia: A peer-to-peer information system based on the XOR metric*, 2002. (Cited on pages 4, 5 and 79.)

[Maymounkov 2002b] P. Maymounkov and D. MaziÃ¨res. *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*. In 1st International Workshop on Peer-to-peer Systems, 2002. (Cited on pages 105 and 107.)

[min ] (Cited on page 5.)

[Muscariello 2011] L. Muscariello, G. Carofiglio and M. Gallo. *Bandwidth and storage sharing performance in information centric networking*. In ACM SIGCOMM workshop on Information-centric networking (ICN '11), 2011. (Cited on page 122.)

[nap ] *Napa-Wine European Project*. `http://napa-wine.eu/`. (Cited on pages 2, 10 and 111.)

[Newman 2001] M. E. J. Newman, S. H. Strogatz and D. J. Watts. *Random graphs with arbitrary degree distributions and their applications*. Phys. Rev. E, vol. 64, page 026118, Jul 2001. (Cited on pages 64 and 65.)

[Nguyen 2008] K. Nguyen, G. Ngo Hoang and H. N. Chan. *Characterizing Chord, Kelips and Tapestry algorithms in P2P streaming applications over wireless network*. In Proc. of International Conference on Communications and Electronics, 2008. (Cited on page 105.)

[Ohm 2004] J.R. Ohm. Multimedia communication technology. Springer, USA, 2004. (Cited on page 133.)

[omn ] *Omnet++ Network Simulator*. http://www.omnetpp.org. (Cited on pages 52 and 129.)

[Piro 2011] G. Piro, L.A. Grieco, G. Boggia, R Fortuna and P. Camarda. *Two-level Downlink Scheduling for Real-Time Multimedia Services in LTE Networks.* In IEEE Trans. Multimedia, to be published, volume 13, pages 1052 – 1065, Oct. 2011. (Cited on page 133.)

[Pouwelse 2005] J. A. Pouwelse, P. Garbacki, D. H. J. Epema and H. J. Sips. *The Bittorrent P2P File-Sharing System: Measurements and Analysis.* In Proc. of International workshop on Peer-To-Peer Systems, 2005. (Cited on page 104.)

[Pujol ] 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 Proc. of ACM SIGCOMM '10. (Cited on page 139.)

[Ratnasamy 2001] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker. *A Scalable Content-Adressable Network.* In ACM SIGCOMM, 2001. (Cited on pages 4, 90, 105 and 108.)

[Ratnasamy 2003] Sylvia Ratnasamy, Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin and Fang Yu. *Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table.* Mobile Networks and Applications, 2003. (Cited on page 5.)

[Rekhter 1995] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP 4).* Internet Engineering Task Force: RFC 1771, 1995. (Cited on page 19.)

[Ren 2008] Dongni Ren, Y.-T. Hillman Li and S.-H. Gary Chan. *On Reducing Mesh Delay for Peer-to-Peer Live Streaming.* In IEEE INFOCOM 2008., 2008. (Cited on page 117.)

[Rossi 2011] D. Rossi and G. Rossini. *Caching performance of content centric networks under multi-path routing (and more).* In Technical report, Telecom ParisTech, 2011. (Cited on pages 130 and 131.)

[Rossi 2012] D. Rossi and G. Rossini. *On sizing CCN content stores by exploiting topological information.* In IEEE INFOCOM, NOMEN Worshop, 2012. (Cited on page 122.)

[Rossini 2012] G. Rossini and D. Rossi. *Large scale simulation of CCN networks.* In In Algotel 2012, 2012. (Cited on pages 122 and 129.)

[Salakhutdinov 2009] R. Salakhutdinov and G. Hinton. *Semantic hashing.* International Journal of Approximate Reasoning, vol. 50, no. 7, pages 969–978, 2009. (Cited on page 90.)

[sec ] *Second life website.* `http://secondlife.com/`. (Cited on page 5.)

[Shafaat 2007] 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. (Cited on page 16.)

[.Shen 2009] X .Shen, H. Yu, J. Buford and M. Akon. Handbook of peer-to-peer networking. Springer Publishing Company, Incorporated, 2009. (Cited on page 107.)

[Shen 2010a] H. Shen, L. Zhao, Z. Li and J. Li. *A DHT-Aided Chunk-Driven Overlay for Scalable and Efficient Peer-to-Peer Live Streaming.* In Proc. of nternational Conference on Parallel Processing, 2010. (Cited on page 106.)

[Shen 2010b] X. Shen, H. Yu, J. Buford and M. Akon, editeurs. Handbook of Peer-to-Peer Networking. Springer-Verlag, 2010. To appear. (Cited on page 6.)

[Stoica 2001] 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. (Cited on pages 3, 79 and 84.)

[Stoica 2003] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek and H. Balakrishnan. *Chord: a scalable peer-to-peer lookup protocol for internet applications.* IEEE/ACM Trans. Netw., vol. 11, no. 1, 2003. (Cited on pages 105 and 107.)

[syn ] *Synapse OverSim implementation webpage.* http://www-sop.inria.fr/teams/lognet/synapseV2.0/. (Cited on page 54.)

[TEL ] *The European Library.* http://www.theeuropeanlibrary.org/. (Cited on page 92.)

[tor ] *The Onion Router project webpage.* (Cited on page 5.)

[Tortelli 2011] M. Tortelli, I. Cianci, L. A. Grieco, G. Boggia and P. Camarda. *A fairness analysis of content centric networks.* In Proc. of Int. Conf. on Network of the Future, NOF, Paris, France, Nov. 2011. (Cited on page 122.)

[Tortelli 2012] M. Tortelli, L. A. Grieco and G. Boggia. *CCN Forwarding Engine Based on Bloom Filters.* In Proc. of ACM Int. Conf. on Future Internet Technologies, CFI, Seoul, Korea, Sep. 2012. (Cited on page 122.)

[Trunfio 2007] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov and S. Haridi. *Peer-to-Peer resource discovery in Grids: Models and systems.* Future Generation Computer Systems, 2007. (Cited on page 5.)

[une 2005] *Recommendations for coordination of digitization of cultural heritage in South-Eastern Europe, Conclusions of the Regional Meeting on Digitization of Cultural Heritage.* Review of the National Center for Digitization, 2005. http://elib.mi.sanu.ac.rs/files/journals/ncd/7/ncd07002.pdf. (Cited on page 92.)

[USy ] *Unstructured Synapse GitHub Repository.* http://www.github.com/barravi/Synapse-Unstructured. (Cited on pages 67 and 68.)

[Varvello ] M. Varvello, C. Diot and E. Biersack. *A Walkable Kademlia network for virtual worlds.* In Proc. of IPTPS '09. (Cited on page 5.)

[Varvello 2011] M. Varvello, I. Rimac, U. Lee, L. Greenwald and V. Hilt. *On the design of content-centric MANETs.* In Int. Conf. on Wireless On-Demand Network Systems and Services, (WONS), Jan. 2011. (Cited on page 122.)

[Viger 2005] Fabien Viger and Matthieu Latapy. *Efficient and Simple Generation of Random Simple Connected Graphs with Prescribed Degree Sequence.* In Computing and Combinatorics, volume 3595 of *Lecture Notes in Computer Science*, pages 440–449. Springer Berlin / Heidelberg, 2005. (Cited on page 69.)

[web 2007] *A weblog about FRBR: Functional Requirements for Bibliographic Records*, 2007. `www.frbr.org`. (Cited on page 92.)

[Wiegand 2003] T. Wiegand, G.J. Sullivan, G. Bjontegaard and A. Luthra. *Overview of the H.264/AVC video coding standard.* IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 7, pages 560 –576, Jul. 2003. (Cited on pages 125 and 130.)

[Xiao 2008] X. Xiao, Y. Shi and Y. Gao. *On Optimal Scheduling for Layered Video Streaming in Heterogeneous Peer-to-Peer Networks.* In Proc. of the 16th annual ACM international conference on Multimedia, 2008. (Cited on page 104.)

[Xu 2003a] Z. Xu, M. Mahalingam and M. Karlsson. *Turning Heterogeneity into an Advantage in Overlay Routing.* In INFOCOM, 2003. (Cited on page 20.)

[Xu 2003b] Z. Xu, R. Min and Y. Hu. *HIERAS: A DHT Based Hierarchical P2P Routing Algorithm.* In ICPP, 2003. (Cited on pages 7 and 16.)

[Xu 2012] H. Xu, Z. Chen, R. Chen and J. Cao. *Live Streaming with Content Centric Networking.* In Proc. 3rd Int. Conf. on Networking and Distributed Computing, Hangzhou, China, 2012., 2012. (Cited on page 122.)

[yac ] *Yacy distributed search engine.* `http://www.yacy.net`. (Cited on page 5.)

[Yang 2003] X. Yang. *NIRA: a New Internet Routing Architecture.* SIGCOMM Comput. Commun. Rev., vol. 33, no. 4, pages 301–312, 2003. (Cited on page 19.)

[Yiu 2007] W.-P. K. Yiu, X. Jin and S.-H. G. Chan. *VMesh: Distributed Segment Storage for Peer-to-Peer Interactive Video Streaming.* IEEE Journal on Selected Areas in Communications, vol. 25, no. 9, pages 1717–1731, 2007. (Cited on page 105.)

[You 2012] Wei You, B. Mathieu, P. Truong, J. Peltier and G. Simon. *DiPIT: A Distributed Bloom-Filter Based PIT Table for CCN Nodes.* In Computer

Communications and Networks (ICCCN), 2012 21st International Conference on, pages 1 –7, 30 2012-aug. 2 2012. (Cited on page 122.)

[Z. Ognjanović 2009] B. Marinković Z. Ognjanović T. Butigan-Vučaj. Ncd recommendation for the national standard for describing digitized heritage in serbia. Springer, 2009. (Cited on pages 92 and 93.)

[Zaharia 2006] M. Zaharia and S. Keshav. *Gossip-based search selection in hybrid peer-to-peer networks.* In Proc. of 5th International Workshop on Peer-to-Peer Systems, 2006. (Cited on page 106.)

[Zhang 2010] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J.D. Thornot, D.K. Smatters, B. Zhang, G. Tsudik, D. Krioukov, D. Massey, C. Papadopulos, T. Abdelzaher, L. Wang, P. Crowley and Yeh E. *Named Data Networking (NDN) Project.* PARC Technical Report TR-2010-02, Oct. 2010. (Cited on page 5.)

[Zhou 2003] S. Zhou, G. R Ganger and P. Steenkiste. *Balancing Locality and Randomness in DHTs.* Rapport technique, 2003. (Cited on page 20.)

[Zhu 2003] D. Zhu, M. Gritter and D. R. Cheriton. *Feedback Based Routing.* SIGCOMM Comput. Commun. Rev., vol. 33, no. 1, pages 71–76, 2003. (Cited on page 19.)

# From key-based to content-based routing: system interconnection and video streaming applications

**Abstract:** Key-based and Content-based routing are a novel approach to message routing amongst nodes in a network, where the destination and path of a message packet is determined not by a well-defined endpoint, but from the message content itself. As such, it has been successfully applied in several networking fields, such as Structured Overlay Networks (SON) and Content-Centric Networks (CCN). Structured overlays are a type of peer-to-peer applications where nodes - members of a logical network - are responsible for a segment of the addressing space on top of which resources are mapped, and messages are routed to the responsible node depending on a key, which often derived by the resource ID; hence the name Key Based Routing (KBR). Content-Centric Networks, on the other hand, are a new routing protocol for the Future Internet, where the concept of nodes is dropped altogether and a routing scheme exclusively based on the desired resource identifier is enforced. The scope of this thesis is twofold: on the one side, we explore the topic of overlay network interconnection and cooperation, and propose an architecture capable of allowing several heterogeneous overlay networks, with different topologies and different routing schemes, to interact, thanks to a lightweight infrastructure consisting of co-located nodes. Through the use of simulations and real-world deployment, we show how this solution is scalable and how it facilitates quasi-exhaustive routing, with even a relatively low number of well-connected co-located nodes. To address the problem of scaling network design to millions of nodes, we propose a mathematical model capable of deriving basic performance figures for an interconnected system. Furthermore, we present two application examples that could greatly benefit from such an architecture. On the other side, we investigate a little further into the capabilities of content-based routing outside of its "comfort zone": first, we analyze the improvement that a SON could bring to a peer-to-peer real-time video streaming system (P2P-TV), in terms of chunk loss and Quality of Experience. Then, we move the approach to a fully content-based domain, implementing the P2P-TV solution on top of Content-Centric Networks.

**Keywords:** Content-based routing, peer-to-peer, overlay networks, network interoperability, overlay protocols, video streaming, p2p-tv.