EFPL Technical Report IC/2005/023

The essence of P2P: A reference architecture for overlay networks*

Karl Aberer[†], Luc Onana Alima[‡], Ali Ghodsi[§], Sarunas Girdzijauskas[†], Manfred Hauswirth[†], Seif Haridi[§]

[§]Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

[¶]Université de Mons-Hainaut (UMH)
B-7000 Mons, Belgique

[||]Swedish Institute of Computer Science (KTH)
S-164 29 Kista, Sweden

Abstract

The success of the P2P idea has created a huge diversity of approaches, among which overlay networks, for example, Gnutella, Kazaa, Chord, Pastry, Tapestry, P-Grid, or DKS, have received specific attention from both developers and researchers. A wide variety of algorithms, data structures, and architectures have been proposed. The terminologies and abstractions used, however, have become quite inconsistent since the P2P paradigm has attracted people from many different communities, e.g., networking, databases, distributed systems, graph theory, complexity theory, biology, etc. In this paper we propose a reference model for overlay networks which is capable of modeling different approaches in this domain in a generic manner. It is intended to allow researchers and users to assess the properties of concrete systems, to establish a common vocabulary for scientific discussion, to facilitate the qualitative comparison of the systems, and to serve as the basis for defining a standardized API to make overlay networks interoperable.

1 Introduction

P2P is not a new paradigm and in fact has already been applied in the original Internet's design, for example, in basic Internet routing or in applications such as Usenet News. What is new, however, is its broad application to all system layers and to new application domains. Most prominently, the P2P approach has been applied for resource location by building so-called *overlay networks*, such as Gnutella, Freenet, Pastry, P-Grid, or DKS, on top of a physical network. Basically all these overlay networks provide a *resource location service* supporting application specific identifiers. On top of these resource location service different application services can be realized, such as data management (search, insert, update, etc.). In principle, distributed application services could also use directly the physical networking layer for managing their resources, but using an overlay network has the advantage of supporting application-specific identifiers and semantic routing, and offers the possibility to provide additional, generic services for supporting network maintenance, authentication, trust, etc., all of which would be very hard to integrate into and support at the networking layer. The introduction of overlay networks and self-management at the service-level are probably the essential innovations of P2P systems.

A wide range of algorithms, structures, and architectures for overlay networks have been proposed already, integrating knowledge from many different communities, such as networking, distributed systems, databases, graph theory, agent systems, complex sytems, etc. The terminologies and abstractions used, however, are quite inconsistent, which makes it very hard to assess and compare different approaches. Only a few relevant attempts to remedy this situation exist so far. For example, JXTA [9] defines a 3-layer architecture (kernel, services, application), XML-based communication protocols, and basic abstractions, such as peer groups, pipes, and advertisements. JXTA intends to provide a uniform programming platform for P2P applications and facilitate interoperability. It provides well-structured APIs and a clear separation of concerns in its architecture but does not mean to describe the structural and functional properties of overlay networks as we do in this paper. Our work and JXTA are thus complementary.

Dabek et al. [6] propose a common API for structured overlays, basically for CAN [14], Chord [16], Pastry [15], and Tapestry[18]. The API only takes into account structured overlays and the used abstraction are at a very low level (C programming interface level), so that using it as a general architecture for modeling overlay networks is not possible.

In this paper we thus propose a reference model for overlay networks which is capable of modeling all existing approaches in this domain. We focus on *decentralized overlay networks* such as Gnutella [5], Freenet [4], CAN [14], Chord [16], P-Grid [1], DKS [3], etc., as this class is the most relevant one. From a modeling point of view, *centralized P2P systems*, such as Napster, are simply client-server architectures where the participants can directly communicate after a discovery phase (similar to a DNS name lookup and then contacting a web server, for example). *Hierarchi-*

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935.

cal P2P systems such as Kazaa, basically consist of a decentralized overlay network of super-peers for locating resources that are used by the normal peers. Thus these system can be modeled by our proposed model with an additional client-server step when contacting a super-peer.

Our model is intended to support the assessment of system properties, establishes a common vocabulary, facilitates the qualitative comparison of the systems, and can serve as the basis for defining a standardized API to make overlay networks interoperable. The major contributions of our model are (1) a conceptual model capturing the concept of embedding a graph into a virtual identifier space, which is fundamental for all overlay networks and (2) a well-defined peer architecture comprising user-level interfaces for applications wanting to use the overlay, interfaces for intra-network communication among homogeneous peers, and interfaces for cooperation among heterogeneous overlay networks.

2 Conceptual Model for Overlay Networks

In any overlay network a group of peers $\mathcal{P}$ provides access to a set of resources $\mathcal{R}$ by mapping $\mathcal{P}$ and $\mathcal{R}$ to an (application-specific) identifier space $\mathcal{I}$ using two functions $F_P : \mathcal{P} \to \mathcal{I}$ and $F_R : \mathcal{R} \to \mathcal{I}$. These mappings establish an association of resources to peers using a closeness metric on the identifier space. To enable access from any peer to any resource a logical network is built, i.e. a graph is embedded into the identifier space. These basic concepts of overlay networks are depicted in Figure 1.

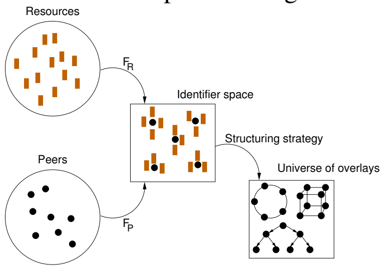

Figure 1. Overlay network design decisions

Each specific overlay network is characterized by the decisions made on the following six key design aspects.

1. choice of an identifier space
2. mapping of resources and peers to the identifier space
3. management of the identifier space by the peers
4. graph embedding (structure of logical network)
5. routing strategy
6. maintenance strategy

In taking the design decisions the following key requirements for overlay networks are addressed:

Efficiency: Routing should incur a minimum number of overlay hops (with minimum "physical" distance) and the bandwidth (messages) for constructing and maintaining the overlay should be kept minimal.

Scalability: The concept of scalability includes many aspects. We focus on numerical scalability, i.e., very large numbers of participating peers without significant performance degradation.

Self-organization: The lack of centralized control and frequent changes in the set of participating peers requires a certain degree of self-organization, i.e., in the presence of *churn* the overlay network should self-reconfigure itself towards stable configurations. This is a *stabilization* requirement as external intervention typically is not possible.

Fault-tolerance: Participating nodes and network links can fail at any time. Still all resources should be accessible from all peers. This is typically achieved by some form of redundancy. This is also a *stabilization* requirement for the same reason as above. Fault-tolerance implies that the *partial failure* property of distributed systems [17] is satisfied, i.e., even if parts of the overlay network cease operation, the overlay network should still provides an acceptable service.

Cooperation: Overlay networks depend on the cooperation of the participants, i.e., they have to trust that the peers they interact with behave properly in respect to routing, exchange of index information, quality of service, etc.

In the following we will provide detailed formal specifications for these key design concepts of overlay networks and discuss the issues related to the requirements listed above.

2.1 Choice of Identifier Space

A central decision in designing an overlay network is the selection of the *virtual identifier space* $\mathcal{I}$ which has to possess some *closeness metric* $d : \mathcal{I} \times \mathcal{I} \to \mathbf{R}$, where $\mathbf{R}$ denotes the set of real numbers. d must satisfy properties 1–3 below and if possible should satisfy properties 4–5.

$$\forall x, y \in \mathcal{I} \;:\; d(x,y) \geq 0 \tag{1}$$
$$\forall x \in \mathcal{I} \;:\; d(x,x) = 0 \tag{2}$$
$$\forall x, y \in \mathcal{I} \;:\; d(x,y) = 0 \;\Rightarrow\; x = y \tag{3}$$
$$\forall x, y \in \mathcal{I} \;:\; d(x,y) = d(y,x) \tag{4}$$
$$\forall x, y, z \in \mathcal{I} \;:\; d(x,z) \leq d(x,y) + d(y,z) \tag{5}$$

If d satisfies all the five properties then $(\mathcal{I}, d)$ is a metric space. However, in many cases only the first three properties will be satisfied. In this case we call $(\mathcal{I}, d)$ a *pseudo-metric space*.

The choice of the virtual identifier space is important for several reasons:

- *Addressing:* The identifier space plays the role of an address space used for identifying resources in the overlay network. Each peer and resource in an overlay network receives a virtual identifier taken from $\mathcal{I}$ (explicitly or implicitly).
- *Scalability:* To support very large systems, $\mathcal{I}$ has to be very large. Through a mapping F_P each peer with a physical address in $\mathcal{P}$ is assigned a virtual identifier from $\mathcal{I}$. This is an application of the well-known principle of indirection for achieving numerical scalability.
- *Location-independence:* The virtual identifier space allows peers to communicate which each other irrespective of their actual physical location. This addresses physical address changes and enables mobility.
- *Clustering of resources with peers:* The closeness metric d enables the clustering of resources with peers based on proximity. This is discussed in detail in Section 2.3.

- *Message routing:* Virtual identifiers and the closeness metric d are essential for realizing efficient routing.
- *Preservation of application semantics:* As virtual identifiers can be defined in an application-specific way, application semantics, for example, "proximity" of resources (clustering), can be preserved.

Examples: CAN [14] uses a Euclidean space with virtual identifiers being coordinates in this space. The distance function d is the Euclidean distance. P-Grid [1] uses a prefix-preserving hash function on strings, i.e., $\forall s_1, s_2 : s_1 < s_2 \Rightarrow h(s_1) < h(s_2)$ ($<$ denotes lexicographic order). Identifiers in P-Grid are bit strings and d is defined as (for a k-bit identifier a and an l-bit identifier b): $d(a,b) = min(|\sum_{i=1}^{k} a_i 2^{-i} - \sum_{i=1}^{l} b_i 2^{-i}| \ , \ 1 - |\sum_{i=1}^{k} a_i 2^{-i} - \sum_{i=1}^{l} b_i 2^{-i}|)$ while in Chord and DKS the identifier space is a subset of the natural numbers of size N and $d(x,y) = (y - x) \ mod \, N$.

2.2 Mapping to the Identifier Space

The mapping $F_P : \mathcal{P} \to \mathcal{I}$ associates peers with a unique virtual identifier from $\mathcal{I}$. Different approaches can be distinguished by the properties of the chosen functions F_P:

- *Completeness:* F_P may be complete or partial. When F_P is partial, peers might (temporarily) not be associated with an identifier.
- *Morphism:* If no replication (for fault-tolerance) is required, F_P will be one-to-one (injective), i.e., $\forall p, q \in \mathcal{P} : p \neq q \Rightarrow F_P(p) \neq F_P(q)$. However, the more typical case is that the system uses replication and the mapping is not injective.
- *Dynamicity:* F_P can be either statically defined, e.g., by its physical address or other unique attributes, or dynamically changing over time. In order to simplify our notations, in the following we will focus on the structural aspects and will not explicitly represent time-dependency in our notations.

Additionally, F_P may satisfy certain distributional properties, for example, that the range of values of F_P follows a certain distribution in space $\mathcal{I}$, e.g., uniform. Such properties may then be exploited, for example, for load balancing. The properties F_P satisfies will be denoted as $\mathcal{C}_{F_P}$ in the following.

The mapping $F_R : \mathcal{P} \to \mathcal{I}$ associates resources with identifiers from $\mathcal{I}$. The choice of this mapping can be critical for the application using the resources. Typically "semantic closeness" of resources, e.g. resources frequently jointly requested, can be translated into closeness of identifiers. Thus the possibility of using application-specific identifiers is taking advantage of this. If the resources should be identified uniquely, F_R has to be injective. The distribution of identifiers generated by F_R has an important impact on the load-balancing properties of the overlay network embedded into the space $\mathcal{I}$.

Examples: A standard example for F_P and F_R is a uniform hashing function as, e.g., used by Chord [16]. This will generate a uniform distribution of peers on the identifier space and implicitly provides load-balancing as also the resource identifiers are uniformly distributed. However, clustering of information will not be possible and thus higher-level search predicates such as range queries will be expensive to process. P-Grid's mapping functions on the other hand supports clustering but thus requires an explicit load-balancing strategy.

2.3 Management of the Identifier Space

At any point in time, $\mathcal{I}$ is managed by the set of current peers $\mathcal{P}$. The responsibility for peers for specific identifiers is captured by a function $\mathcal{M} : \mathcal{I} \to 2^{\mathcal{P}}$, which associates with each identifier of a resource r, $i = F_R(r) \in \mathcal{I}$, the set of peers that are managing r. Through $\mathcal{M}$, each peer p is assigned *responsibility* for the set $\mathcal{M}^{-1}(p)$ of identifiers. Locating a resource r corresponds to finding a peer in $\mathcal{M}(F_R(r))$. The lookup operation of overlay networks typically provides an implementation of $\mathcal{M}$ through routing. We may identify various basic properties for the function $\mathcal{M}$.

- *Completeness:* $\mathcal{M}$ may be complete or partial. When $\mathcal{M}$ is incomplete, identifiers might (temporarily) not be associated with a peer. Typically the mapping will be complete, such that each point of the identifier space is under responsibility of some peers, i.e., $\forall i \in \mathcal{I} : \exists p \in \mathcal{P} : p \in \mathcal{M}(i)$
- *Cardinality:* To provide fault-tolerance, $\mathcal{M}$ typically contains more than one element, i.e., a set of peers is responsible for managing each identifier.
- *$\mathcal{M}$ induced by proximity:* A standard way to specify $\mathcal{M}$ is that identifiers are associated with their closest peers, i.e. $p \in \mathcal{M}(i) \Rightarrow d(F_P(p), i) = min_{q \in \mathcal{P}} d(F_P(q), i)$.
- *Dynamicity:* $\mathcal{M}$ typically changes dynamically as the set of peers and their mapping to the identifier space changes.
- *Uniformity of replication:* The cardinality of $\mathcal{M}$ (which corresponds to the degree of replication) may be constant or uniformly distributed to ensure comparable availability of resources. Non-uniform distributions can be used to adapt the availability of resources to application requirements, e.g., popularity of resources.

In the following, $\mathcal{C}_{\mathcal{M}}$ denotes the properties $\mathcal{M}$ satisfies.

Examples: In Chord a peer with virtual identifier a is responsible for the interval $(predecessor(a), a]$, i.e., $min\{F_P(a) \ominus F_P(b) | b \in \mathcal{P}\} \oplus F_P(b)$. In P-Grid a peer with a k-bit path a is responsible for all identifiers in the interval $[\sum_{i=1}^{k} a_i 2^{-i} \ , \ 2^{-k} + \sum_{i=1}^{k} a_i 2^{-i})$.

2.4 Graph Embedding

An overlay network can be modeled as a *directed graph*, $G = (\mathcal{P}, \mathcal{E})$, where $\mathcal{P}$ denotes the set of vertices (i.e., peers) and $\mathcal{E}$ denotes the set of edges. Due to the dynamics in overlay networks, G is time-dependent, but as before we will not explicitly denote this. By virtue of this graph we define a *neighborhood* relationship $\mathcal{N} : \mathcal{P} \to 2^{\mathcal{P}}$, such that for a given peer p, $\mathcal{N}(p)$ is the set of peers with which peer

p maintains a connection, i.e., there is a directed edge (p, q) in $\mathcal{E}$ for $q \in \mathcal{N}(p)$.

The properties of the overlay network relate to properties of the directed graph generated by $\mathcal{N}$ and to the properties of the embedding of the graph into the (pseudo-) metric space $(\mathcal{I}, d)$. Purely structural properties of the graph can be further distinguished into local and global properties, i.e., whether they relate to local characteristics of graph nodes or to global characteristics of the graph. Typical global properties of the graph are the following:

- *Uniqueness:* For deterministic systems, e.g., Chord, DKS, for a given set $\mathcal{P}$ and mapping F_P only one valid network $\mathcal{N}$ exists. In randomized systems such as P-Grid and randomized Chord multiple valid $\mathcal{N}$ are possible.
- *Graph diameter:* A small diameter provides lower bounds on the latency of routing in the network.
- *Connectivity*: Some overlay network approaches may require that the overlay graph is connected at any time.
- *Distributional properties:* These are typically distributional properties of node degrees. A frequently occurring class of graphs are power-law graphs [12]. Other distributional properties relate to the clustering coefficient of the graph.

Typical local properties of the graph include:

- *Minimal out-degree:* This property is beneficial to ensure fault-tolerance, when many neighbors fail.
- *Maximal out-degree:* This property is relevant for ensuring bounded maintenance cost for connections to other peers.
- *Distributional properties of in-degree:* These are relevant for load balancing in the message forwarding.

More complex properties refer to relationships of the graph structure to the distance function. These relationships are tightly intertwined with the strategy for efficient routing in an overlay network. Typical examples of such constraints are:

- *Local connectivity:* This property ensures that peers are connected to some specific subset of their immediate neighbors. An example of such a requirement for a given peer p would be $\forall q \in \mathcal{P} : d(F_P(p), F_P(q)) < d_{min} \Rightarrow q \in \mathcal{N}(p)$
- *Long-range connectivity:* Many overlay network designs are structurally similar to small-world graphs as introduced by Kleinberg [11]. These graphs are constructed such that long range connections satisfy the condition $P[q \in \mathcal{N}(p)] \propto \frac{1}{d(F_P(p), F_P(q))^{-d}}$, where d is the dimensionality of the identifier space. Many overlay networks satisfy more strict variations of this condition.

The properties $\mathcal{N}$ satisfies are denoted by $\mathcal{C}_{\mathcal{N}}$ in the following. At this point we are able to completely characterize the structural aspects of overlay networks by the following definition.

Definition. The structure of an overlay network $O \in \mathcal{O}$ for a set of peers $\mathcal{P}$ is given by $O = (\mathcal{I}, d, F_P, \mathcal{C}_{F_P}, \mathcal{M}, \mathcal{C}_{\mathcal{M}}, \mathcal{N}, \mathcal{C}_{\mathcal{N}})$.

2.5 Routing Strategy

The basic service an overlay network provides is to route a request for an identifier i to a peer p_r responsible for it, i.e., $p_r \in \mathcal{M}(i)$. Routing is a distributed process using the overlay network. We model it by asynchronous message passing: $route(p, i, m)$ forwards a message m to a peer p responsible for i. A routing strategy can be described by a potentially non-deterministic function $\mathcal{R} : \mathcal{P} \times \mathcal{I} \to 2^P$, which selects at a given peer p with neighborhood $\mathcal{N}(p)$ for a target identifier i the (set of) next peers $\mathcal{R}(p, i) \in \mathcal{N}(p)$, to which the message is forwarded. In structured overlay networks typically routing is greedy, i.e., $d(i, F_P(q)) < d(i, F_P(p))$ for $q \in \mathcal{R}(p, i)$. Some systems satisfy weaker conditions, e.g., in Pastry, $d(i, F_P(\mathcal{R}(p, i))) \leq d(i, F_P(p)$. In unstructured overlay networks the set $\mathcal{R}(p, i)$ may contain several peers.

Properties of routing algorithms are characterized by their associated cost measures, such as latency, number of hops, and probability of successful routing. Given a routing algorithm together with an overlay network structure the properties regarding the expected usage of the peers' resources can be analyzed.

2.6 Maintenance Strategy

Participation of peers in an overlay network dynamically changes over time. Each peer can freely decide to join or leave an overlay network at any time. These changes, referred to as *churn* in the literature, can happen quite frequently. To maintain the structural integrity of an overlay network a *maintenance strategy* is required, which compensates for changes to the network structure due to peers going offline or failure of network connections.

In all overlay networks, joining the network is done explicitly by a join operation, whereas leaving typically is implicit as peers may simply go offline or crash or their network connection may drop. Regardless whether peers leave gracefully or not, changes in the participation in an overlay network typically require the application of a maintenance strategy. Aside from access control aspects, i.e., who is allowed to participate, this basically requires to repair routing tables which have been invalidated due to churn, i.e., to maintain the connectivity of the underlying graph [7]. Maintenance strategies can be classified [2] into *proactive correction* (PC) using periodic probing or heartbeats to repair inconsistencies, and *reactive mechanisms*, with the subcategories *correction on use* (CoU), e.g., P-Grid and DKS, *correction on failure* (CoF), e.g., P-Grid, and correction on change (CoC), e.g., Chord.

The practical usability of an overlay network critically depends on the efficiency of the maintenance strategy. The goal is to maintain a "sufficient" level of consistency while minimizing effort. Since a dynamically evolving overlay network on top of a dynamically changing physical network is a complex dynamical system, the goal is to arrive at a stable dynamic equilibrium for a variety of conditions while guaranteeing successful routing.

2.7 Other Properties

There are a number of further properties which we can only briefly mention here due to space limitations.

Constraints, such as those introduced in the previous sections, can be guaranteed at different levels of strictness: If the constraints are valid all the time, they are *invariants* of the system; if the constraints hold *eventually*, they may be satisfied after *self-stabilization* of the system induced by changes to the systems state; if the constraints hold *probabilistically*, they are satisfied with a specific probability either all the time or eventually.

By taking into account the *physical characteristics* of peers, such as their network location, their storage capacity, etc., additional properties can be specified which are in particular useful to obtain insights and control over the performance characteristics of the overlay network, for example, efficiency of routing and reliability of the network. An important example of such a property is *locality of routing.* A possible formulation of such a property is a constraint on the stretch introduced by the overlay network, i.e., that the physical distance of the path traversed to reach a node does not exceed the distance of the shortest physical path by a given stretch factor.

3 Reference architecture

From an application-oriented perspective, any middleware technology–and we see P2P systems and specifically overlay networks as a form of middleware–should provide powerful and easy to use abstractions that hide implementation details as much as possible from the user/implementer while offering enough control and access options to actually meet application requirements. Additionally, the abstractions should be defined in a way that the concrete infrastructure implementing the middleware functionality can be replaced without requiring to rewrite code.

Given these goals, we see P2P systems based on overlay networks as layered systems as depicted in in Figure 2 (for a single node). From a user's perspective a P2P system facilitates to realize a specific application by sharing resources with other users and using services provided by the P2P layer. One particularly important example of such a service is P2P data storage, which allows to insert, search, and access data items. This service as well as the applications take advantage of the basic resource location service provided by the P2P basic layer that implements the overlay network.

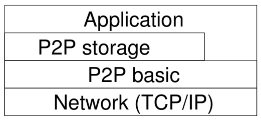

Figure 2. Layered architecture view

This simple layered architecture supports separation of concern between the application layer, the generic services of a P2P system and the basic overlay network of a P2P system. This facilitates to replace a specific implementation of a P2P system, or selected services and layers, that an application is using by alternative implementations. In order to support this form of modularity it is important to provide a standardized specification of the interfaces among the layers. In Figure 3 we provide a class diagram that provides the core of such an interface specification. It is based on the conceptual model we have introduced in Section 2.

Overlay networks are based on the embedding of a graph into an *Identifier Space* which provides a closeness metric. Each *Peer* is mapped into this space, i.e., it is assigned an *Identifier* from the virtual identifier space, which defines its current position in this space and (indirectly) the subset of identifiers the peer is responsible for as described in Sections 2.2 and 2.3. Note that a peer's position can change over time. How the partitioning of the identifier space is done, i.e., how a node is assigned a coordinate and responsibility, is subject to the specific overlay approach. A *Peer* is uniquely identified by an immutable name (*immutableName)* and maintains a neighborhood (*neighbors*), i.e., references to other peers (*PeerReference*) for forwarding. Each *PeerReference* includes the referenced peer's immutable name, its position in the identifier space, i.e., responsibility, and physical network address (IP address or symbolic name). As this information changes over time, each peer has to apply a maintenance strategy as discussed in Section 2.6 to have a consistent view (depending on the specific overlay network). The number of neighbors a peer maintains and the strategy how neighbors are selected is defined by the *Constraints* of the overlay network which depend on properties of the identifier-resource and identifier-peer association strategies, the graph embedded in the identifier space, and its constraints, etc.

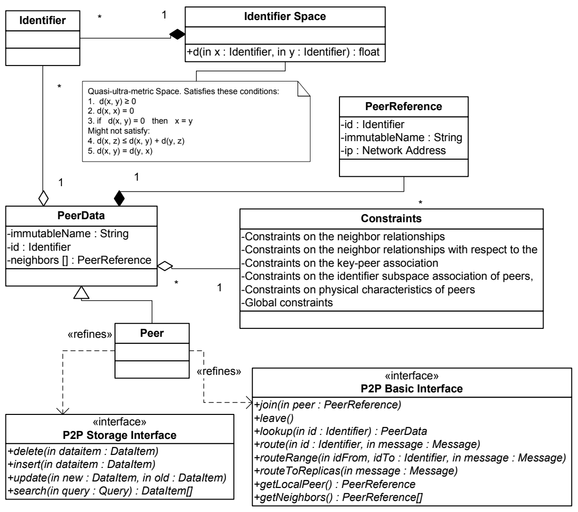
Identifier
*
1
Identifier Space
+d(in x : Identifier, in y : Identifier) : float
*
Quasi-ultra-metric Space. Satisfies these conditions:
1. d(x, y) ≥ 0
2. d(x, x) = 0
3. if d(x, y) = 0 then x = y
Might not satisfy:
4. d(x, z) ≤ d(x, y) + d(y, z)
5. d(x, y) = d(y, x)
PeerReference
-id : Identifier
-immutableName : String
-ip : Network Address
1
1
*
PeerData
-immutableName : String
-id : Identifier
-neighbors [] : PeerReference
Constraints
-Constraints on the neighbor relationships
-Constraints on the neighbor relationships with respect to the
-Constraints on the key-peer association
-Constraints on the identifier subspace association of peers,
-Constraints on physical characteristics of peers
-Global constraints
*
1
«refines»
Peer
«refines»
«interface»
P2P Storage Interface
+delete(in dataitem : DataItem)
+insert(in dataitem : DataItem)
+update(in new : DataItem, in old : DataItem)
+search(in query : Query) : DataItem[]
«interface»
P2P Basic Interface
+join(in peer : PeerReference)
+leave()
+lookup(in id : Identifier) : PeerData
+route(in id : Identifier, in message : Message)
+routeRange(in idFrom, idTo : Identifier, in message : Message)
+routeToReplicas(in message : Message)
+getLocalPeer() : PeerReference
+getNeighbors() : PeerReference[]

Figure 3. Conceptual decomposition

As shown in in Figure 2, we distinguish two layers of functionality. The basic layer (*P2P Basic Interface*) provides the low-level operations which the overlay needs to be able to function. Its main functionalities, besides the mandatory *join* and optional *leave* operations, are the *lookup* and *route* operations. The *lookup* function allows an application to find a peer by its identifier to be able to directly communicate with it (point-to-point), for example, for transferring data items. The *route* operation, which lookup typically builds on, allows the user to send a message to any peer responsible for a given identifier. A message can contain any data specified by the application, for example, the data to be stored by the peer or a synchronization request among replicas. *routeToReplicas* propagates a given message to the set of peers responsible for the same identifier. *getLocalPeer* returns the administrative information about the local peer and *getNeighbors* provides the list of neighbors of a peer, i.e., its routing table information.

The storage layer (*P2P Storage Interface*) builds on these functionalities and provides the typical data management functionalities of inserting, updating, deleting, and querying data, that made the P2P paradigm popular. The resources affected by the functions are specified via the *DataItem* abstraction that includes the resource's data and the application-specific key(s) to be used by the storage layer to generate a corresponding identifier, i.e., map the data item to its position in the identifier space. This can then be used by the basic layer to find the responsible peer(s) and perform the requested operation. The *DataItem* set returned by *search* includes both the application-specific keys and the the identifiers of the found resources.

We would like to emphasize that Figure 3 provides a *minimal* model, i.e., it provides what we identified as the minimal common denominator for different overlay network approaches. All parts of the architecture can be (and in fact are) extended by concrete systems. For example, each system will typically have more structured message types. For example, in Gnutella, as one of the simplest systems, a *join* operation would mean the issuing of a *Ping* message which has a simple structure holding a descriptor ID (to prevent loops in the routing), a payload descriptor, a time-to-live counter, a hops counter and a field defining the length of the payload. Yet, extensions of our model are intuitive and simple: A concrete system can basically "subclass" and "extend" any of the components in Figure 3.

4 Interoperability

Up to now we have introduced a conceptual model and abstract interfaces to capture the specific properties of a given overlay network approach. In practice, multiple overlay networks will co-exist simultaneously in a physical network, which raises issues of managing multiple overlay networks and interoperability.

We consider an overlay network as a group of peers $\mathcal{P}$ that share the same specification of their specific overlay network mechanisms. The sharing of this specification is a problem of group management and can be done either explicitly or implicitly.

With an *explicit management* explicit group identifiers $\mathcal{G}$ (e.g. URIs) are used to identify an overlay network and are bound to a specific type of overlay network by a mapping $\mathcal{T} : \mathcal{G} \to \mathcal{O}$, which associates the identifier with a specification of an overlay network. We consider this as providing the overlay network with a type (or schema in database parlance). Thus every peer joining a group $g \in \mathcal{G}$ obtains the associated type information and adheres to the specification. The issue of non-complying peers is related to security and trust which we cannot elaborate further here. As a consequence, joining an overlay network would only be possible if the joining peer uses the same group identifier as the peers of the network.

With *implicit management* a group of peers is considered as participating in the same overlay network if they use the same overlay network specification. Thus there is no global knowledge on the existence of a specific overlay network, but the network results from the cooperation of peers using the same specification. Thus when joining, a peer obtains/shares the specification with the peer to which it joins.

Another interesting aspect of group management in an overlay network is the *degree of coupling*. In *tightly coupled* overlay network the overlay graph is at any time connected. This implies that such an overlay network has to be initiated by a single peer (that could, for example, determine the identifier and specification of the network properties, when explicit group management is used). Chord is an example of a tightly coupled overlay network. In *loosely coupled* overlay networks different overlay graphs based on the same specification (e.g., using implicit group management) can evolve, merge, or split. Gnutella and P-Grid are examples of loosely coupled overlay networks.

The approach to implicitly manage groups of peers participating in the same overlay network suggests a more general view of how groups of peers constructing overlay networks may work together. In order to interact, it is in fact not necessary that the type of overlay network is exactly the same, but it may be sufficient that the specifications are compatible. This approach can be observed for some practical overlays systems, such as Gnutella. Multiple versions of overlay protocols can work together, and different peers may use different policies, e.g., with respect to network connectivity.

For characterizing the possibilities of interoperability among peers participating in different overlay networks O_1 and O_2, we can systematically compare the specifications of the networks. We assume that at the level of protocols, O_1 and O_2 are compatible by following the API defined in Section 3 and using compatible protocol messages. This is a purely syntactic agreement. The classification of interoperability follows the concepts described in Section 2 and we can distinguish the following levels of structural interoperability:

- *Compatible Identifiers:* The identifier spaces $\mathcal{I}_1$ and $\mathcal{I}_2$ are the same or can be related to each other by applying a transformation. Then for identifiers in $i \in \mathcal{I}_1 \cap \mathcal{I}_2$, peers from both O_1 and O_2 can route messages to the resources identified by i. Routing would be processed independently in O_1 and O_2. Thus peers can play the role of gateways among different overlay networks.
- *Compatible Identifier Spaces:* If additionally the distance functions (possibly after applying a transformation) are compatible, peers from O_1 may use peers from O_2 (and vice versa) and their knowledge on neighbors to integrate them into their own routing ta-

bles.

- *Compatible Structures:* If additionally the structural constraints of two overlay networks are in a subsumption relationship, i.e., one of the overlay networks is more constrained but compatible with the more general overlay network, peers of the more constrained network may participate as peers in the less constrained network by adopting the routing and maintenance algorithms of the less constrained network.

An important open issue, when exploiting these forms of structural interoperability, are the effects on the performance of the routing and maintenance mechanisms and the impact on certain structural properties of the overlay networks, such as distributional properties. These questions are closely related to the study of overlay networks built by peers with highly heterogeneous resources, a topic which has been studied only to a very limited degree so far.

5 Validation of the reference architecture

In this section we will briefly describe key aspects of a representative set of overlay networks–Chord [16], DKS [3], P-Grid [1], Pastry [15], Freenet [4], and Gnutella [5]–in terms of our architecture to demonstrate its validity. Additionally, we provide a brief qualitative comparison of the systems.

5.1 Identifier Space

The identifier spaces are very similar for all logarithmic-style overlay networks (P-Grid, Chord, Pastry, etc.). In these approaches identifiers are chosen from an alphabet with radix b, e.g., $b = 2$ for P-Grid and Chord, $b = 16$ for Pastry. Some of them limit the identifier length, e.g., Pastry uses 128-bit, Chord and DKS use 160-bit length identifiers, whereas in P-Grid identifiers can be of arbitrary length. A similar distance function is shared by all of these overlays, though there are some subtle differences. In P-Grid and Pastry the distance $d(u,v)$ of two identifiers u (of length k) and v (of length l) is $min(|\sum_{i=1}^{k} u_i b^{-i} - \sum_{i=1}^{l} v_i b^{-i}|\ ,\ 1 - |\sum_{i=1}^{k} u_i b^{-i} - \sum_{i=1}^{l} v_i b^{-i}|)$. Note that in Pastry's case $k = l$), and $d(u,v)$ is symmetric as $d(u,v) = d(v,u)$. For example, in P-Grid, $d(\text{“0000”}, \text{“10”}) = d(\text{“10”}, \text{“0000”}) = 0.5$.

The identifier space in Chord is not symmetric, i.e., $d(u,v) \neq d(v,u)$. $d(u,v)$ can be defined as $((\sum_{i=1}^{k} v_i 2^{-i} - \sum_{i=1}^{k} u_i 2^{-i}) + 1)\ mod\ 1$. Thus $d(\text{“001”}, \text{“111”}) = 0.75$, but $d(\text{“111”}, \text{“001”}) = 0.25$.

In Freenet the situation is slightly different. Due to the way Freenet identifies nodes, it uses an r-dimensional 160-bit identifier space. r depends on the data items a peer stores, but usually $r = 50$. $d(u,v)$ between two Freenet peers is the Euclidean distance in this multidimensional space.

5.2 Mapping to the Identifier Space

Mapping of peers: The key difference among the overlays with respect to this mapping is whether the virtual identifier is assigned to a peer randomly or the peer adopts the identifier depending on environment conditions, e.g., depending on the data a peer and its neighboring peers store. In Chord and Pastry the virtual identifier is generated using some random function and assigned to a peer upon joining the overlay and remains stable. In DKS and SkipNet identifiers can be mapped order-preserving based on their domain name, e.g., lexicographic ordering, to ensure that nodes in the same organizational domain are logically close in the identifier space.

Most of the logarithmic-style overlay approaches like Chord, Pastry, or P-Grid have a one-dimensional identifier space. In Chord 160-bit and in Pastry 128-bit identifiers are assigned by hashing the node's IP address using SHA-1. In contrast, in P-Grid each peer initially is responsible for the whole identifier space and has an empty identifier which grows bit by bit in the lifetime of the peer depending on which other peers it encounters and what type of data they and the peer itself store. Similarly in Freenet, each node assigns itself an identifier vector of size r, consisting of r 160-bit elements representing the r identifiers of data items the peer stores. Additionally, the identifier of a Freenet peer changes during its lifetime depending on the queries it handles. Thus the identifiers in P-Grid and Freenet dynamically change, whereas in Pastry and Chord they are static.

Mapping of resources: Mapping of resources (data items) is done similarly to mapping peers. Usually it is done by hashing a data key, e.g., the filename, with SHA-1 (Chord, Pastry, Freenet). While this implicitly distributes the assigned identifiers uniformly in the identifier space and thus provides a simple load-balancing mechanism, it destroys the semantics of keys, e.g., their application-specific clustering, which can be exploited to provide efficient data access. To prevent this, P-Grid, for example, uses a prefix-preserving hash function, i.e., $u < v \Rightarrow h(u) < h(v)$. This has advantages in query processing but requires an additional and more complex load-balancing strategy. It is crucial that this mapping of resources is deterministic, static, and globally known.

5.3 Management of Identifier Space

In P-Grid each peer is responsible for resource identifiers that share the largest common prefix with the peer's identifier, i.e., a resource identifier is managed by the peer with the closest identifier in terms of P-Grid's distance function. For example, peer “0011” is responsible for resource identifier “001110101”, if no peer with a longer common prefix exists. The situation in Freenet is very similar: Each peer is responsible for the resource identifiers which are numerically closest to one of the peer's elements in its vector identifier. Also in Pastry a similar condition applies. Data items are managed by the peer with the closest identifier. For example, identifier “2A83” will be managed by peer “2A84” if no peers “2A82” and “2A83” exist. As Chord's and DKS's identifier spaces are asymmetric, the situation is slightly different. A peer is responsible for all identifiers in the interval between its own identifier and the identifier of its predecessor on the ring. In all these approaches the responsibility of a peer may dynamically change due to arrivals or departures of peers in the overlay.

5.4 Graph Embedding

It has already been shown that peers cooperating in Freenet evolve the graph into a small-world graph. For logarithmic-style overlay approaches, [8] shows that these approaches form graphs according to Kleinberg's small-world principles [11]. It is proven that such graphs belong to the special class of "routing efficient" small-world networks where decentralized greedy search algorithms provide the best performance. Therefore conceptually all of these approaches build similar small-world graphs with certain constraints for each case. E.g., in the logarithmic-style overlay case each peer u views the identifier space as partitioned in $\log N$ partitions where each partition is b times bigger than the previous one (b is the radix of identifier alphabet). The routing table of u in such systems contains $\log_b N$ links to some nodes from each partition. In Chord's case the chosen node will be the one with the smallest identifier of the given partition, while Pastry and P-Grid use any random node in the partition, which is a more relaxed constraint.

5.5 Gnutella

Gnutella's underlying paradigm has major conceptual differences compared to the structured overlay systems described above. Despite its simplicity, the description of Gnutella is not trivial. On the surface, it may seem that Gnutella has no identifier space and no mappings are done. But then it would not possible to describe Gnutella as a graph embedded in an identifier space. Nevertheless, we can assume that peers in Gnutella exist in an Euclidean identifier space R and each peer assigns itself a random identifier $i \in R$. Then we assume that each peer is responsible for the whole identifier space and therefore it is not necessary to map resources to the identifier space and each peer additionally chooses four random neighbors. Given such conditions and Gnutella's routing and maintenance strategies, the peers form a small-world graph in the identifier space. As this graph has a very low diameter of approximately $\log N$, constrained flooding for search works efficiently in terms of latency. This shows that also Gnutella fits our conceptual model of overlay networks.

6 Related Work

Although a large number of overlay networks have been devised, only very few works on unifying architectures exist. The closest ones, JXTA [9] and Dabek et al. [6] have already been discussed in the introduction. In a recent work the structural properties of a subclass of overlay networks have been characterized by using algebraic methods (Cayley graphs) [13]. This work is complementary as it could be used to formulate more specific constraints on the structure of overlay networks within our architectural framework. In [10] classifications for structured overlay networks, e.g., deterministic and randomized networks, are introduced. These classifications correspond to different constraints that we can capture in our conceptual model. To best of our knowledge, no other proposals for an overlay network reference architecture exist.

7 Conclusions

Based on a stringent analysis of current overlay networks, we discussed and formally described the key design aspects in this domain. We then used our assessments to define a reference architecture for overlay networks specifically addressing API and interoperability aspects. To validate the correctness and general applicability of our approach we applied it in modeling a representative set of overlay networks. Our reference architecture establishes a standardized vocabulary and facilitates the assessment of properties of overlays for qualitative comparison and may serve as the basis for the definition of a standardized API.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *CoopIS*, 2001.

[2] K. Aberer, A. Datta, and M. Hauswirth. Route maintenance overheads in DHT overlays. In *WDAS*, 2004.

[3] L. O. Alima, S. El-Ansary, P. Brand, and S. Haridi. DKS(N,k,f): A Family of Low Communication, Scalable and Fault -Tolerant Infrastructures for P2P Applications. In *CC-GRID*, 2003.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, 2001.

[5] *The Gnutella Protocol Specification v0.4 (Document Revision 1.2)*, June 15 2001. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

[6] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *IPTPS*, 2003.

[7] A. Ghodsi, L. O. Alima, and S. Haridi. Low-Bandwidth Topology Maintenance for Robustness in Structured Overlay Networks. In *HICSS*, 2005.

[8] S. Girdzijauskas, A. Datta, and K. Aberer. On Small-World Graphs in Non-uniformly Distributed Key Spaces. In *NetDB, Tokyo, Japan*, 2005.

[9] L. Gong. JXTA: A Network Programming Environment. *IEEE Internet Computing*, 5(3):88–95, May/June 2001.

[10] K. Gummadi, R. Gummadi, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of the ACM SIGCOMM*, 2003.

[11] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *STOC*, 2000.

[12] M. Mitzenmacher. A Brief History of Generative Models for Power Law and Lognormal Distributions. Draft manuscript. http://www.eecs.harvard.edu/~michaelm/postscripts/tempim1.ps, 2005.

[13] C. Qu, W. Nejdl, and M. Kriesell. Cayley DHTs - A Group-Theoretic Framework for Analyzing DHTs Based on Cayley Graphs. In *ISPA*, 2004.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, 2001.

[15] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[16] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[17] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[18] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.