# The Gamut of Bootstrapping Mechanisms for Structured Overlay Networks

Anwitaman Datta

**Abstract** Structured overlays are an important primitive in building various peer-to-peer (P2P) systems, and is used for various functions including address independent end-to-end routing, managing multicast groups, indexing of content in a decentralized environment and P2P storage, among others. While they operate in a decentralized manner, and the self-stabilizing mechanisms to maintain the overlays are also decentralized, bootstrapping structured overlays have traditionally assumed implicit centralization and/or coordination. In this chapter, we provide a survey of different flavors of structured overlay construction mechanisms – including quasi-sequential mechanisms which are predominantly in use, followed by parallelized approaches, and finally looking into how two isolated overlay can be merged, which is key to decentralized bootstrapping.

## 1 Introduction

In recent years the concept of structured overlays[1] has attracted a lot of attention because of its potential to become a generic substrate for internet scale applications – used for applications as diverse as locating resources in a wide area network in a decentralized manner, address independent robust and flexible (group) communication – e.g., application layer multicast and internet indirection infrastructure and content distribution network to name a few.

The basic function of the structured overlay is to act as a decentralized index. To that end, for each resource, a globally unique identifier (called the key) is generated using some function suitable to the applications that are supposed to use the index.

Anwitaman Datta
NTU, Nanyang Avenue, Singapore, e-mail: anwitaman@ntu.edu.sg

[1] A special class of structured overlays are the *distributed hash tables* (DHTs), where the keys are generated from the resources (name or content) using uniform hashing, e.g., SHA-1 (Secure Hash Algorithm [17]).

X. Shen et al. (eds.), *Handbook of Peer-to-Peer Networking,*
DOI 10.1007/978-0-387-09751-0_11, 

The codomain (loosely speaking, range) of this function is called the key-space. For example, the key-space may be the unit interval $[0,1]$ or an unit circle $[0,1)$, so that the keys can be any real number between 0 and 1. The key-value pair is stored at peers responsible for the particular key. Efficient search of keys based on decentralized routing helps the applications to access the resource itself in absence of central coordination or global knowledge.

**Definition 1.** Structured overlay networks comprise of the following principal ingredients:
(i) Partitioning of the key-space (say an interval or circle representing the real number between the range $[0,1]$) among peers, so that each peer is responsible for a specific key space partition. By being responsible, we mean that a peer responsible for a particular key-space partition should have all the resources (or pointers) which are mapped into keys which are in the respective key-space partition.[2]
(ii) A graph embedding/topology among these partitions (or peers) which ensures full connectivity of the partitions, desirably even under churn (peer membership dynamics), so that any partition can be reached from any partition to any other – reliably and preferably, efficiently.
(iii) A routing algorithm which enables the traversal of messages (query forwarding), in order to complete specific search requests (for keys).

**Definition 2.** A structured overlay network thus needs to meet two goals to be functionally correct:
(i) *Correctness of routing*: Starting from any peer, it should be possible to reach the correct peer(s) which are responsible for a specific resource (key).
(ii) *Correctness and completeness of keys-to-peers binding*: Any and all peers responsible for a particular key-space partition should have all the corresponding keys/values.

Various applications can use transparently the (dynamic) binding between peers and their corresponding key-space partitions as provided by the overlay for resource discovery and communication purposes in a wide area network.

One of the most important and distinguishing aspect of structured overlays is the peers' interconnection – the topology/geometry of the network.

How this topology is established in a dynamic setting, and whether it achieves some other properties (like – proximity and low stretch exploiting information from the underlying networking layer, load-balancing, security against various attacks, etcetera.) and how the invariants of the topology maintained over time in presence of membership dynamics and attacks are some of the most interesting questions that have been investigated in the P2P research community in these last years.

[2] It is also possible that keys are not strictly associated with a specific peer and instead have a looser coupling. For example, in Freenet [9] and FuzzyNet[15], this association of keys to peers can be thought to be in a best effort fashion, such that instead of choosing the peer which is globally the closest to a key, the locally closest peer is delegated the responsibility of the key. Such systems are also called semi-structured overlays.

In this article, we focus on the issue of how these topologies can be established (bootstrapped) in a dynamic setting while also meeting some other desirable properties like load-balancing. We will survey different kinds of bootstrapping mechanisms, including traditional quasi-sequential approaches, as well as subsequent parallelized approaches, and also looking at how two isolated overlays can be merged to build a larger network, paving way for decentralized boot strapping.

The existing literature presenting various bootstrapping mechanisms vary in rigor and details. To keep the presentation uniform as well as accessible to the general audience, we provide only a high level summary of the concepts.

## 2 A Taxonomy of Structured Overlay Topologies

The specific details of the topologies is crucial to the exposition of how these topologies can be achieved. We briefly look into some of the important structured overlay topologies, particularly the *ring* and *tree* topologies. These are not necessarily optimal in the sense of achieving the smallest routing table size or smallest diameter, however have proven to be practical because of their overall characteristics. They have moderately small average routing table sizes which provide good resilience at reasonable maintenance cost, small diameter, good degree-distribution (necessary for congestion-free and load-balanced routing) and flexibility to deal with different kind of workloads, and last but not the least, they are also relatively simple. The complexity of the topology plays an important role in a peer-to-peer setting, where the topology invariants need to be established and maintained without global knowledge and coordination in presence of potentially large scale in terms of both peer population as well as high membership dynamics.

### *2.1 Ring*

The ring based topology was pioneered in the context of overlays in the Chord [27] network. Chord uses SHA-1 based consistent hashing to generate an $m$-bit identifier for each peer $p$, which is mapped onto a circular identifier space (key-space).

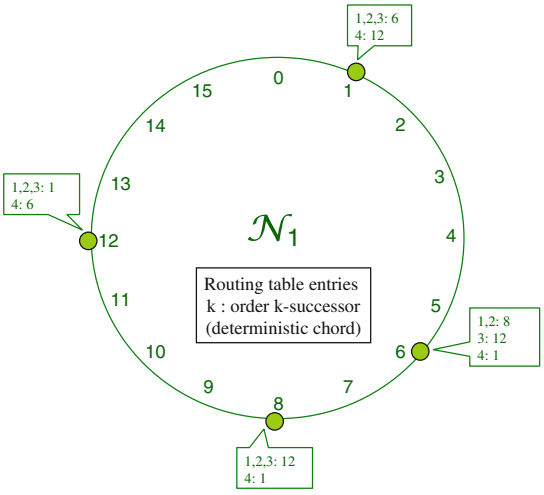
Irrespective of how the peers' identifiers are generated in a ring based topology, what is essential is that the peer identifiers are distinct. Similarly, unique keys are generated corresponding to each resource. Each *key* on the key-space is mapped to the peer with the least identifier greater or equal to the *key*, and this peer is called the *key*'s successor. Thus to say, this peer is responsible for the corresponding resource.

What is relevant for our study is how keys from the key-space are associated with some peer(s) and how the peers are interconnected (in a ring) and communicate among themselves.

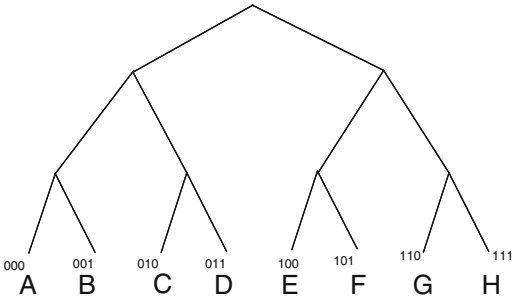
**Definition 3.** A ring network is (1) weakly stable if, for all nodes $p$, we have $predecessor(successor(p)) = p$; (2.a) strongly stable if, in addition, there exists

no peer $s$ on the identifier space where $p < s < q$ where $successor(p) = q$; and (2.b) loopy if it is weakly but not strongly stable.

Condition (2.a) that there exists no peer $s$ on the identifier space where $p < s < q$ if $p$ and $q$ know each other as mutual successor and predecessor determines the correctness of the ring structure. Figure 1a shows one such consistent ring structure (peer's position in the ring and its routing table). The order-1 successor known also just as "successor" of each peer is the peer closest (clock-wise) on the key-space.




(a) A consistent ring (Chord) network




(b) A tree based (P-Grid) network. The actual topology has no hierarchy as shown in Figure 2.

**Fig. 1** Some structured overlay topologies

If at any time such a $s$ joins the system, the successor and predecessor information needs to be corrected at each of $p$, $q$ and $s$. Maintaining the ring is basically to maintain the correctness of successors for all peers – this in turn provides the func-

tional correctness of the overlay routing – i.e., successor peer for any identifier *key* can be reached from any other peer in the system (by traversing the ring). For redundancy, $f_s$ consecutive successors of each peer are typically maintained, so that the ring invariant is violated only when all $f_s$ consecutive peers of any peer depart the system even before a ring maintenance mechanism like Chord's self-stabilization algorithm can react and repopulate with the correct successor entries.

In addition to the successor/predecessor information, each peer maintains routing information to some other distant peers in order to reduce the communication cost and latency.

It is the way these long ranges are chosen which differ in many ring topology networks and has no critical impact on the functional correctness of the overlay. Distance in such ring based topologies is generally measured in terms of the absolute difference of the two concerned points on the key-space, but other metrics can as well be used. For the real topology, devoid of the artificial distance metrics, the long ranges are essentially to halve the number of peers (the "true" distance on a ring traversed sequentially) between the current peer and the destination peer [14].

Explicitly or implicitly, most variants of the ring topology exploit this fact and reduce the distance geometrically – either deterministically or probabilistically. The original Chord proposal advocated the deterministic use of the successor of the identifier $(p+2^{k-1})$ modulo $2^m$ as an order-$k$ successor of peer $p$ or a finger table entry. Many other variants for choosing the long range links exist – e.g., randomized choice from the interval $[p+2^{k-1}, p+2^k)$ or exploiting small-world [20] topology or emulating skip graphs.

The maintenance of the ring (strong stability) is critical for functional correctness of the routing process in ring based topologies. The self-stabilization mechanisms proposed in the original Chord proposal [27] exhaustively deals with the maintenance of the ring, and all other ring based topologies rely on similar mechanisms. It has been shown that the ring topology has better static resilience than other topologies because of the greater flexibility to choose both routing table entries to instantiate the overlay, as well as to choose from multiple routes to forward a query at run time.

### 2.1.1 Ring Self-Stabilization Highlights

The ring invariant is typically violated when new peers join the network, or existing ones leave it. If such events occur simultaneously at disjoint parts of the ring, the ring invariant can easily be reestablished using local interactions among the affected peers. Note that these events do not lead to a loopy state of the network.

Apart looking into the simple violations of the ring invariant which are relatively easily solved, the original Chord proposal (technical report version) also provides mechanisms to arrive at a strongly stable network starting from a loopy network (whichsoever reason such a loopy state is reached). We summarize the results of stabilizing a loopy network here.

Any connected ring network with $N$ peers becomes strongly stable within $O(N^2)$ rounds of strong stabilization if no new membership changes occur in the system. Starting from an arbitrary connected state with successor lists of length $O(logN)$ if the failures rate is such that at most $N/2$ nodes fail in $\Omega(logN)$ steps then, whp, in $O(N^3)$ rounds, the network is strongly stable.

## *2.2 Tree*

Arguably the earliest approach to locate objects in a distributed environment – the PRR scheme proposed by Plaxton et al. [25] used a tree structure where searches were forwarded based on longest prefix matching. Tapestry [28], Pastry [26] (also uses the ring as a fall-back mechanism) and P-Grid [1] shown in Fig. 1b among others uses similar prefix resolution in order to forward search operations, and has the tree topology. The leaf-nodes of the tree represent the key-space partitions (peers). The (maximum) distance between these partitions when the query is resolved based on prefix is then the height of the common subtree. Kademlia [22] resembles the tree structure and peers have the same routing choices as other tree-based networks. Despite having the same topology, Kademlia routing uses the XOR distance between the peer identifiers (essentially the binary string representing the node's path in the tree) instead of resolving common prefix.

Note that this also exemplifies the essential orthogonality of the topology itself from the routing strategy – the same graph connectivity may be explored based on different routing schemes, and thus defined as separate ingredients of a structured overlay network in Definition 1.

### 2.2.1 The P-Grid Overlay

Some of the concepts in this article will be illustrated using examples of the P-Grid network, thus we next provide a formal description of P-Grid. Figure 1b shows the tree abstraction and Fig. 2 shows one possible instance of peers' connections.

P-Grid divides the key-space in mutually exclusive partitions so that the partitions may be represented as a prefix-free set $\Pi \subseteq \{0,1\}^*$. Stored data items are identified by keys in $\mathcal{K} \subseteq \{0,1\}^*$. We assume that all keys have length that is at least the maximal length of the elements in $\Pi$, i.e.,

$$\min_{k \in \mathcal{K}} |k| \geq \max_{\pi \in \Pi} |\pi| = \pi_{max}$$

Each key belongs uniquely to one partition because of the fact that the partitions are mutually exclusive, that is, different elements in $\Pi$ are not in a prefix relationship, and thus define a radix-exchange trie.

$$\pi, \pi' \in \Pi \Rightarrow \pi \not\sqsubseteq \pi' \wedge \pi' \not\sqsubseteq \pi$$

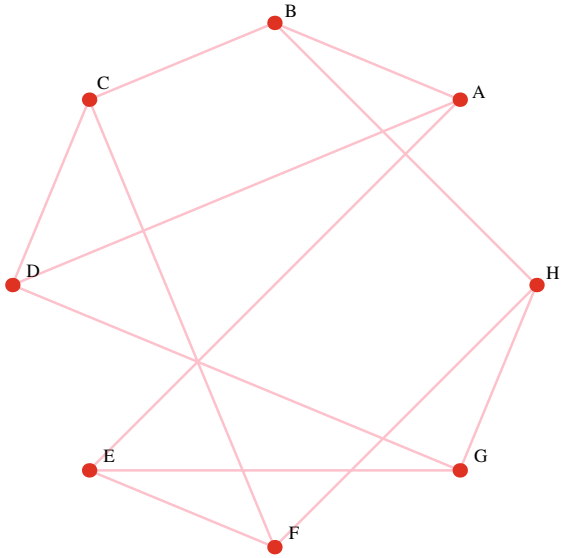



**Fig. 2** The actual P-Grid connectivity graph does not have any hierarchy. The routes are randomly chosen from complimentary sub-trees of Fig. 1b. The basic P-Grid graph is directional, however since each link establishment and maintenance cost is the same, and from the symmetry of the routing choices, the actual P-Grid uses bidirectional routes

where $\pi \subseteq \pi'$ denotes the prefix relationship. These partitions also exhaust the key-space, so to say, the key-space is completely covered by these partitions so that each key belongs to one and only one (because of exclusivity) partition.

In P-Grid each peer $p \in P$ is associated with a leaf of the binary tree, and each leaf has at-least one peer associated to itself. Each leaf corresponds to a binary string $\pi \in \Pi$, also called the *key-space partition*. Thus each peer $p$ is associated with a path $\pi(p)$. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length $l$ a set of references $\rho(p,l)$ to peers $q$ with property $\overline{\pi(p,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string $\pi$ with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level which enables the implementation of prefix routing for efficient search. The whole routing table at peer $p$ is then represented as $\rho(p)$ Moreover, the actual instance of the P-Grid is determined by the randomized choices made at each peer for each level out of a much larger combination of choices. The cost for storing the references and the associated maintenance cost scale as they are bounded by the depth of the underlying binary tree. This also bounds the search time and communication cost.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ the binary key $\kappa(d)$ is calculated using an order-preserving hash function. $\kappa(d)$ has $\pi(p)$ as prefix but it is not excluded that temporarily also other data items are stored at a peer, that is,

the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. Moreover, for fault-tolerance, query load-balancing and hot-spot handling, multiple peers are associated with the same key-space partition (*structural replication*). $\Re(\kappa)$ represents the set of peers replicating the object corresponding to key $\kappa$. Peers additionally also maintain references to peers with the same path, i.e., their replicas $\Re(\pi(p))$, and use epidemic algorithms to maintain replica consistency. Routing in P-Grid is greedy, and based on matching the longest prefix, similar to the PRR scheme [25].

There are many other topologies derived from interconnection networks that have been adapted for P2P settings, but the rest of the chapter will focus on overlays based on the prevalent ring and tree topologies.

## 3 Quasi-Sequential Construction of Overlays

The construction of overlays, from the early days have focussed not only in establishing a logically correct topology, but also on how to ensure that load distribution across the peers is uniform.

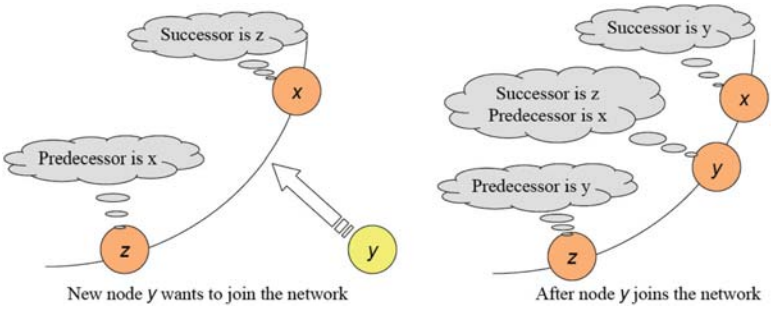



**Fig. 3** A new node joining a Chord network. Nodes $x$ and $z$ which are originally neighbors (predecessor and successor of each other) need to update their local information to register that $y$ is the new neighbor

The original approach to construct most structured overlays assumed an incremental approach, investigating how new nodes can continuously join an existing network, or how the network manages continuous but gradual departures of existing nodes – called churn in the network.

Churn in the network requires reallocation of the part of the key space that a peer is responsible for, as well as rewiring of the connections, for instance in the ring based networks, ensuring that the ring integrity is maintained is most important. Ring integrity is ensured by making sure that nodes maintain the right successor (and predecessor) lists despite churn. To ensure this, it is important to allow only

one node to join between a node and its immediate successor at any given time. Figure 3 depicts such a scenario. Such a scheme makes an implicit assumption that nodes join in the same part of the key-space in a sequence. Of-course nodes can join mutually disjoint parts of the key-space simultaneously. Essentially such a model can thus support quasi-sequential join of peers in the network.

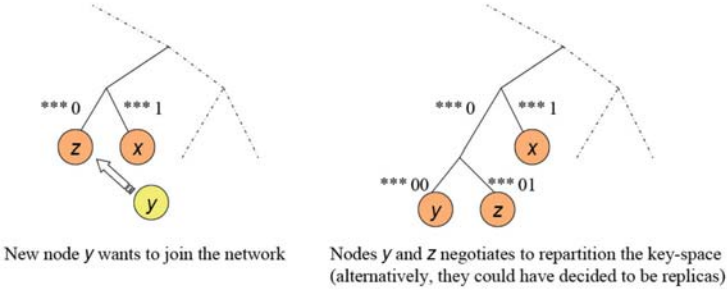



**Fig. 4** A new node $y$ joining a P-Grid network by redividing the responsibility with existing peer $z$. They also need to update their routing tables, so that, for example, if $z$ receives a query with prefix $\cdots 00$ it can forward it to $y$. Other peers, for example $x$ do not need to update any information, and can continue to forward all queries with prefix $\cdots 0$ still to $z$

In the case of an overlay like P-Grid abstracting the tree structure, the sequential overlay construction will require the newly joining peer to negotiate with an existing peer to redivide key-space partition responsibility (as shown in Fig. 4) or alternatively decide to become mutual replicas.[3]

## *3.1 Load-Balancing Considerations*

In order to achieve load-balancing, randomized strategies are often used in such a setting. Each new node joining the system picks a random point uniformly on the key-space, and joins the network by splitting that part of the key-space with whichever peer has been originally responsible for that part of the key-space. Uniformly choosing the part of the key-space to join the network was assumed to evenly partition the key-space. While not highlighted as such, such a randomized strategy coincidentally makes sure that different peers joining at the same time join different parts of the network, doing so in an almost trivial way without any need for global knowledge or coordination.

Uniform random choice of peer's position in the key-space however leads to a relatively high variation in the size of key-space partitions for which individual

[3] Replication is necessary for both load-sharing and fault-tolerance, but has not been shown in this specific example. Structurally replicated P-Grid networks will be shown later in Fig. 12.

peers are responsible for. A lot of work has been done to reduce such variation, and to make the load distribution among peers more uniform. Included among those approaches is a simple but very effective randomized strategy "power of two choices", where each node would choose two (or several) random points on the key-space, and then join the network in the points with most load [12]. Such an approach is a variation of the power of two choices originally proposed by Byers et al. [7] in the context of peer-to-peer systems, where multiple hash values were used to generate multiple keys for a resource, and then stored at the least loaded peer.

For various reasons, explained in the next section, it may be desirable or even necessary to construct an overlay in a parallelized manner, a departure from the original quasi-sequential approach.

## 4 Parallelized Construction of Overlays

Fast construction of structured overlays has been motivated by several reasons. Building such a network rapidly from scratch enables fast recovery from catastrophic failures as well as easy deployment of such a network on demand, which can be used to perform tasks required by more complex distributed systems, as well as eliminate or at least complement complex and expensive overlay maintenance algorithms, allowing for recreation of the whole network instead. From the perspective of considering the structured overlay as a distributed index structure, building an overlay from scratch can be considered to be analogous to (re-)indexing content.

*Further motivation: A data-management perspective*

In standard database systems it is common practice to regularly (re-)index attributes to meet changing requirements and optimize search performance. Structured peer-to-peer overlay networks are increasingly used as an access structure for highly distributed data-oriented applications, such as relational query processing, metadata search or information retrieval [4, 24]. Structured overlays' use was motivated by the presence of certain features that are supported by their design such as scalability, decentralized maintenance, and robustness under network churn. Compared to unstructured overlay networks which are also being proposed for these applications [16, 21], structured overlay networks additionally exhibit much lower bandwidth consumption for search as well as guarantee completeness[4] for search results.

The standard maintenance model for peer-to-peer overlay networks assumes a dynamic group of peers forming a network where peers can join and leave, essentially in a sequential manner, as elaborated in the previous section. In addition proactive or reactive maintenance schemes are used to repair inconsistencies resulting from node and network failures or to re-balance load in order to react to data

[4] In terms of information retrieval terminology, recall = 1.

updates. These approaches to maintenance, that have been extensively studied in the literature, correspond essentially to updating database index structures in reaction to updates.

In data-oriented applications resources may be identified by dynamically changing predicates. Multiple overlay networks can be needed simultaneously, each of them supporting a specific addressing need. One can illustrate these requirements by a typical application case of peer-to-peer information retrieval.

The standard application of structured overlay networks in peer-to-peer information retrieval is the implementation of a distributed inverted file structure for efficient keyword based search. In this scenario, several situations occur, in which the overlay network has to be constructed from scratch:

- A set of documents that is distributed among (a potentially very large number) of peers is identified as holding information pertaining to a common topic. To support efficient retrieval for this specific document collection, a dedicated overlay network implementing inverted file access may have to be set up.
- A new indexing method, for example, a new text extraction function for identifying semantically relevant keywords or phrases, is being used to search a set of semantically related documents distributed among a large set of peers. Since the index keys change as a result of changing the indexing method a new overlay network needs to be constructed to support efficient access.
- Due to updates to a distributed document collection an existing distributed inverted file has become obsolete. This may either result from not maintaining the inverted file during document updates or due to changing characteristics of the global vocabulary and thus changing the indexing strategy (e.g., term selection based on inverse document frequency). Thus a complete reconstruction of the overlay network is required.
- Due to catastrophic network failures the standard maintenance mechanisms no longer can reconstruct a consistent overlay network. Thus the overlay networks needs to be constructed from scratch. Of course, this scenario applies generally in any application, but becomes more probable when multiple overlay networks are deployed in parallel.

In principle a (re-)construction of an overlay network in any of these scenarios can be achieved by the standard maintenance model of sequential node joins and leaves. However, this approach encounters two serious problems:

- The peer community will have to decide on a serialization of the process, e.g., electing a peer to initiate the process. Thus the peer community has to solve a leader election problem, which might turn out to be unsolvable for very large peer populations without making strong assumptions on coordination or limiting peer autonomy.
- Since the process is performed essentially in a serialized manner, it incurs a substantial latency. In particular it does not take any advantage of potential parallelization, which would be a natural approach.

These motivate a fundamentally different approach, that of parallelized overlay construction. Several such mechanisms including [2, 5, 18] have been proposed in the literature, which we summarize next.

## *4.1 Sorting Peer-IDs as a Mechanism to Build a Ring*

To achieve a strongly stable ring (Definition 3) it is necessary that all nodes know the correct immediate neighbors (successor and predecessor). This leads to a sorting of the peers according to their identifiers or responsibility of the parts of the key-space. Achieving this without the global knowledge of which all peers are in the network, and without any central coordination of peers to choose the correct neighbors is a crucial challenge in constructing an overlay. So the problem of constructing a ring based structured overlay essentially boils down to the problem of decentralized sorting of the peers according to their identifiers. Now we discuss one rigorous and one heuristic mechanism to sort peer-IDs.

### 4.1.1 Pairing and Merging Virtual Trees

Angluin et al. [5] considers the problem of constructing a structured overlay as that of creating a linked list of nodes sorted according to their identifiers, which can then be used as the basis for constructing the essential ring of the system. Long range links useful for optimization can be wired subsequently to make the network routing efficient.

They assume that any node in the system can communicate with any other node, once the nodes become aware of each other, which defines a "knowledge graph". Initially, this knowledge graph is assumed to be a weakly connected degree bounded directed (random) graph. That is to say, each node knows a fixed number of random nodes.

The first step in this approach is to pair nodes using a randomized mechanism. Based on the original knowledge graph connectivity, every node probes all potential successors. The recipient of such a probe either accepts (the first received probe) or rejects to be paired. Ideally, after a round of such pairing is finished, the paired nodes can behave as a virtual "supernode" to conduct further pairing. Note that such a pairing of virtual supernodes which are composite of previously paired nodes, require a merging mechanism. The virtual supernodes are maintained as Patricia trees, which facilitates the use of tree merging algorithms, and finally provides a single supernode comprising of all the nodes sorted also as a list.

Figure 5 shows a step of their mechanism as a toy example. Nodes 1 and 7 pair up, and act as a virtual node, as do nodes 5 and 9. Subsequently these two virtual nodes (comprising two nodes each), merge to form a single virtual node. The implicit tree representation allows both efficient mergers, as well as provides readily a tree structure, apart from sorting the peer identifiers.

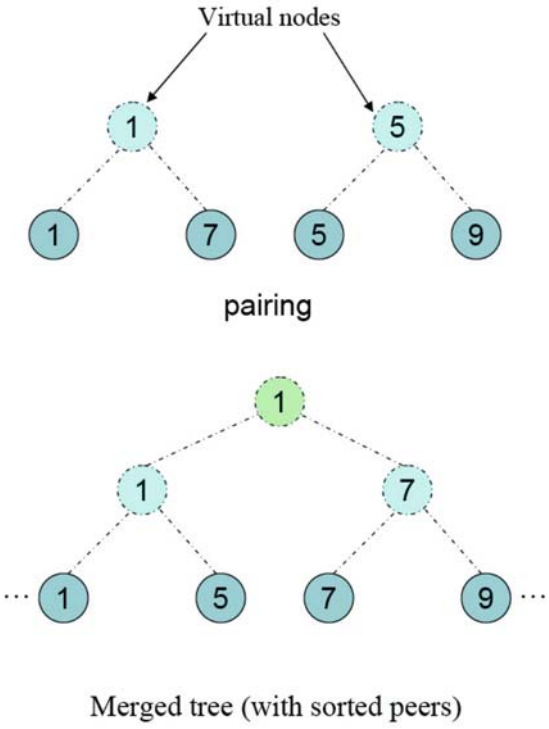



**Fig. 5** Sorting based on iterations of pairing and merging [5]. Once the peer identifiers are sorted, a ring structure can readily be obtained. Essentially, what is necessary in a decentralized setting to achieve such a sorted "linked list" is that all peers know the correct predecessors and successors (multiple entries are useful for fault tolerance)

The main complexities of this approach in a real life scenario (asynchronous setting) are in determining and phasing the pairing rounds and the merging process, and avoiding live-locks. In its original form, the algorithm also could not tolerate any node departure during the overlay construction.

### 4.1.2 Gossip Based Mechanism

Jelasity et al. [18, 19, 23] proposed a gossip based approach to bootstrap overlays. They assume that each node can obtain a random subset of the participating peers. This assumption is similar to the idea of knowledge graph assumed by Angluin et al. [5].

Each node maintains a constant sized leaf-set, comprising of the nearest nodes to itself in both direction – termed as predecessors and successors. Ideally, equal

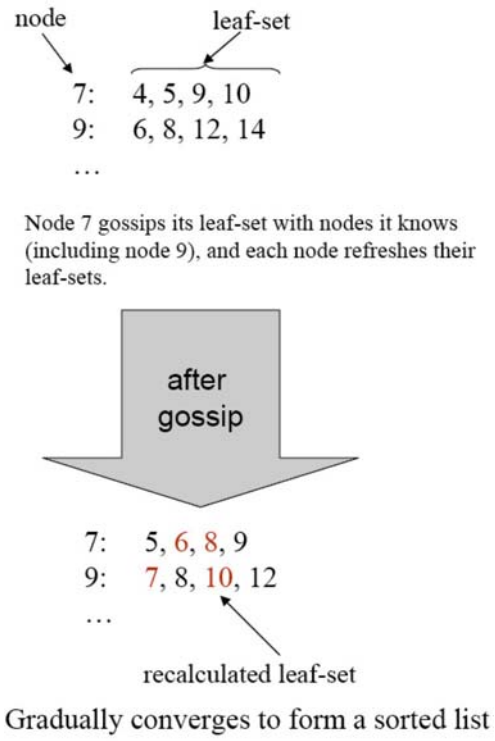



**Fig. 6** Sorting based on iterations of gossips [18, 19, 23].

number of predecessors and successors are maintained, but if a node knows fewer of one kind, then it fills the space with nodes of the other category. Such information originally comes from the random subset of participating peers that each node gets, and eventually is derived from the information that nodes gossip among each other.

The essential idea is to gossip the leaf-set information, and refine it based on the information obtained from other nodes. In the toy example shown in Fig. 6, node 7 originally knows nodes $\{4,5,9,10\}$. Likewise 9 knows $\{6,8,12,14\}$. By gossiping with all the nodes 7 knows, which includes 9, 7 gets to know about 6 and 8, its immediate neighbors in this example. In contrast to the rigorous approach of Angluin et al., the gossip based approach is more a heuristic, in that while there is no formal proof of convergence, simulation based studies show it performs pretty well, and nodes get to establish the ring information with few iterations of the gossip mechanism. This mechanism can also be expected to be robust against node departures and arrivals during the process, and hence may be more practical.

### *4.2 Recursive Proportional Partitioning*

The approach to sort peer identifiers can be considered to be a bottom-up approach, where the peer identifiers are already chosen (often randomly) and thus the way the key-space responsibility should be distributed is pre-determined. A logically top-down approach introduced by Aberer et al. [2] proposes to autonomously and recursively partition the key-space, but in a granularity adaptive to the load-distribution on the key-space (which can be arbitrarily skewed for data-oriented applications).

During the overlay construction process, two types of load balancing problems are dealt with simultaneously – the balancing of storage load among peers under skewed key distributions (i.e., number of keys per partition is balanced) and the balancing of the number of replica peers across key space partitions. The first problem is important to balance workload among peers and is solved by adapting the overlay network structure to the key distribution. The second one is important to guarantee approximately uniform availability of keys in unreliable networks where peers have potentially low availability. This is similar to a classical "balls into bins" scenario, where the key-space partitions are the bins and the peers (replicas) the balls. The extra twist, why existing solutions for balls into bins problems can't however be directly used is that the number of bins (key-space partitions) itself is dynamic.

Similar to Angluin et al. [5] and Jelasity et al. [18, 19, 23], this approach also assumes a (loosely) connected network, so that any node can communicate with any other node, and particularly uses random walkers on this network to find random peers with whom to interact. However in contrast to those approaches, where nodes already have a particular identifier, which determines the part of the key-space partition that node is responsible for, and the task is to sort the nodes to establish a sorted list, necessary to construct a ring, Aberer et al. [2] instead allows the nodes to negotiate among each other to refine the part of the key-space they will be responsible for. It leads to construction of an overlay (P-Grid) which can support prefix based routing.

The bilateral negotiations among the peers is illustrated in Fig. 7. Such interactions can lead to three possible course of actions. Two peers may decide to repartition the part of the key-space they are responsible for, or decide to become mutual replicas, or refer each other to other suitable peers to interact with. Notice that the "repartition or replicate" actions are similar in spirit to the sequential approach described earlier, but now these actions are happening in parallel for different peers, and are iterated several times thanks to the "refer" actions.

The essential idea is that all peers are originally responsible for the whole key-space. Then, they need to find a partner to decide and split responsibility – say, for the prefixes 0 and 1 respectively. The peers also need to keep track to such a peer which is responsible for the other half, so that in future a node responsible for partition 1 can forward all queries for partition 0 to a relevant peer. Thus all peers responsible for the partition 0 can repartition it for 00 and 01, and so on.

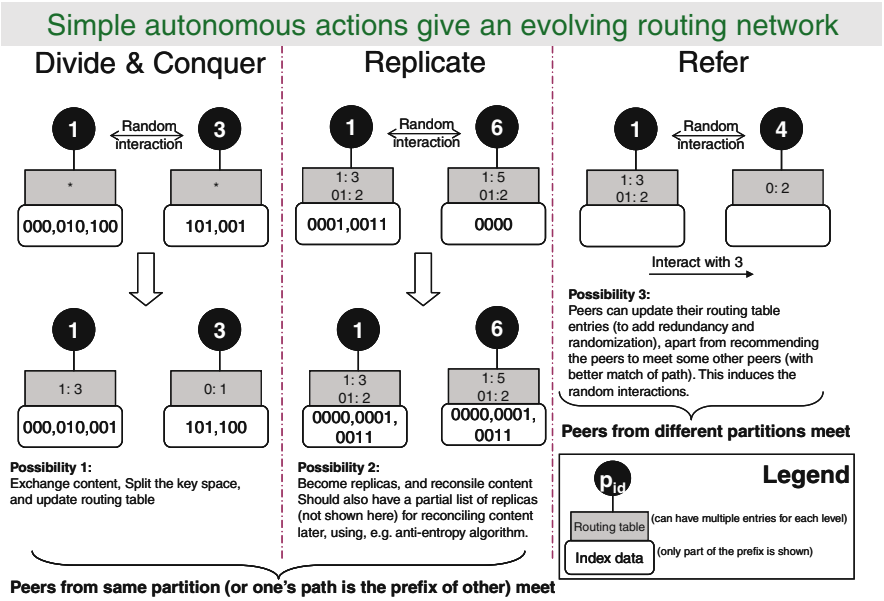



**Fig. 7** Network evolution based on pairwise peer interactions

There are several practical problems in realizing the scheme. This includes the following.

What fraction of peers should choose a specific partition? Ideally this depends on the load in the specific half of the key-space.

For example, for the example in Fig. 8, there is three times more load for the the prefix 0 than the prefix 1, and twice the load for prefix 01 than for 00. So ideally, the peers should evolve into a network, with the partitioning of the key-space and its replication, as shown in the figure.

There are however two practical problems in realizing such an ideal partitioning. Its unrealistic to assume global knowledge like the load-skew. Even if this information were available, without global coordination, it is not possible to achieve partitioning of the network among the peers according to the granularity of the load-distribution.

Aberer et al. [2] provides some heuristics to address these problems. They propose random sampling of a subset of other peers to estimate the load-imbalance at each level of repartitioning, and using this estimate to determine parameters of a randomized algorithm to split the key-space which in expectation follows the desired (estimated) load-skew. Experiments show, that while far from perfect, the heuristics

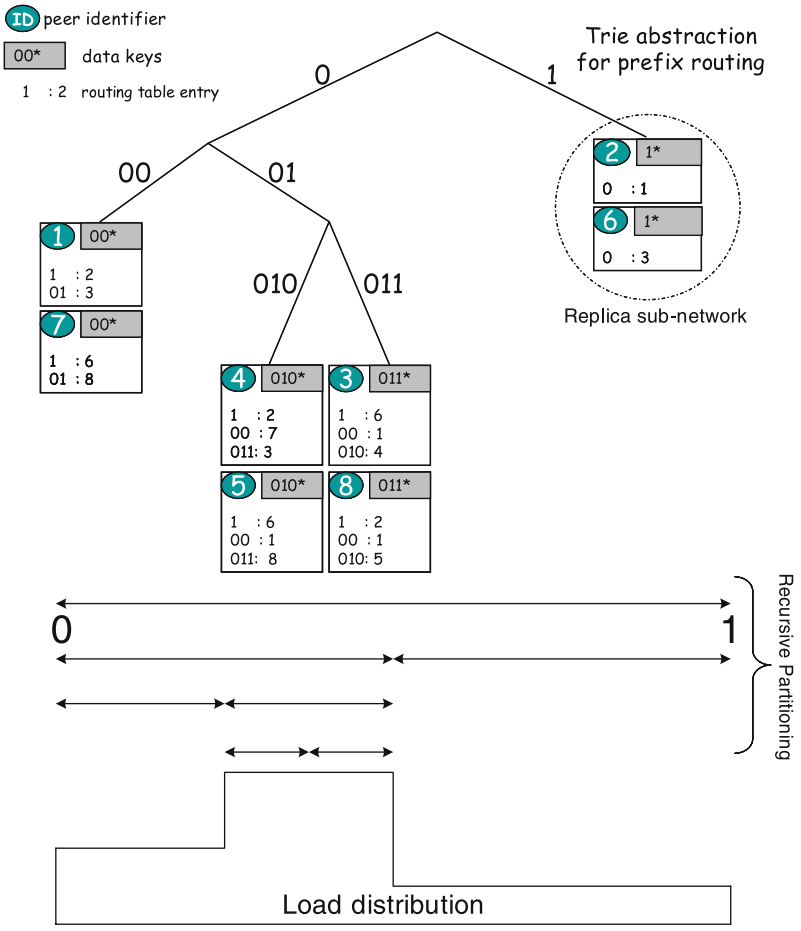



**Fig. 8** P-Grid structure: Key-space is partitioned in a granularity adaptive to load-skew. In this example peers $p_1$ and $p_7$ are structural replicas for the partition for prefix 00. Peer $p_1$ has reference to peer $p_2$ for prefix 1, and to peer $p_3$ for prefix 01. Peer $p_7$ stores the same keys as peer $p_1$ (replicas), however they can and do have different routing table entries. In practice, for each level, each peer will also maintain multiple references primarily in order to have some fault-tolerance. Thus peer $p_1$ would also refer to some of $p_3$, $p_5$ or $p_8$ for the prefix 01

lead to a reasonably well load-balanced network construction. Other maintenance mechanisms [3] are deployed to further improve the quality of the load-balancing, but overall, this approach again achieves the primary objective, that of constructing an overlay from scratch in a parallelized manner, more or less conforming to load-skews to achieve moderate load-balance.

## 5 The Need and Challenges of Merging Two Similar Structured Overlays

All the parallelized construction approaches described above assumed (i) an originally connected network, and that (ii) there is one specific network to join.

Such assumptions implicitly introduces some degree of centralization. What happens if multiple isolated networks are originally constructed?

In the somewhat frantic rush to identify new cool topologies, or more practical problems like dealing with load-balancing and churn, the peer-to-peer research community has until recently ignored a fundamental and realistic problem for structured overlays that any distributed system needs to deal with – that of making two partitions of such a system merge to become one. One can speculate several reasons for such omissions in early structured overlay network research. (i) Merger of isolated overlays is trivially resolved in unstructured overlays, which is where most of the empirical information of P2P research so far has been derived from. (ii) Until recently, there has not been any real structured overlay implementations deployed and hence the problem not identified. (iii) The recent deployments and experiments have typically been under a controlled setting, where some central coordination like the use of a common set of bootstrap nodes has been used with the intention and sufficient coordination to construct only one overlay, making sure that independent and distinct overlays are not created. Moreover, none of these experiments with real implementations looked specifically for, or even accidentally, encounter network partitioning problems.

Apart network partitioning which can lead to the creation of two disjoint overlay networks there is a more likely scenario. It may so happen that disjoint overlay networks (using the same protocols) are formed over time by disjoint group of users. One may imagine that an overlay P2P network caters to a specific interest group from a particular geographic area who participate in an overlay network. At a later time, upon discovering a hitherto unknown group of like-mind users from a different part of the world, who use their own "private" network (using same protocols), these two groups may want to merge their networks in order to benefit from their mutual resources (like content or knowledge). In fact, such isolated overlay networks may result because of initial isolation of groups because of various reasons including geographic, social or administrative – a large company or country, which may originally restrict their users from interacting with outsiders in the overlay, and changes the policy at a later time – or purely because of partitioning of the physical infrastructure.

Structured overlays have often been touted as a generic substrate for other applications and services. Ideally, there will be one or few such universal overlays [8] which will be used by a plethora of other P2P applications. Realizing such an universal service too will need the possibility to merge originally isolated networks. Small overlays can be built independently, which may later be all merged together incrementally into a single overlay network.

One can thus imagine isolated islands of functional overlays catering to their individual participants. Someday, some member from one of these overlays may discover a member from another overlay. The natural thing to do then would be to merge the two originally isolated overlays into a single overlay network. In simple file-sharing networks, the motivation of doing so will be to make accessible content from both the networks to all the users. Similar conclusions can be drawn for various other conceivable applications of overlay networks.

In unstructured overlay networks (like Gnutella), merger of two originally isolated overlays happens trivially. Whichever peers from two originally isolated networks come in contact with each other need to establish mutual neighborhood relationship, and then onwards just need to forward/route messages to each other as they do with all other neighbors (Fig. 9). That's all! Likewise, hierarchical (super-peer based) unstructured overlays also merge together trivially. This is because no peer has any specific responsibility and can potentially be responsible for any and all resources in the network.

One approach to deal with multiple structured overlay networks is to let them continue to exist and operate autonomously, while allowing cross-network communication. This is typically achieved in a hierarchical fashion, where the original

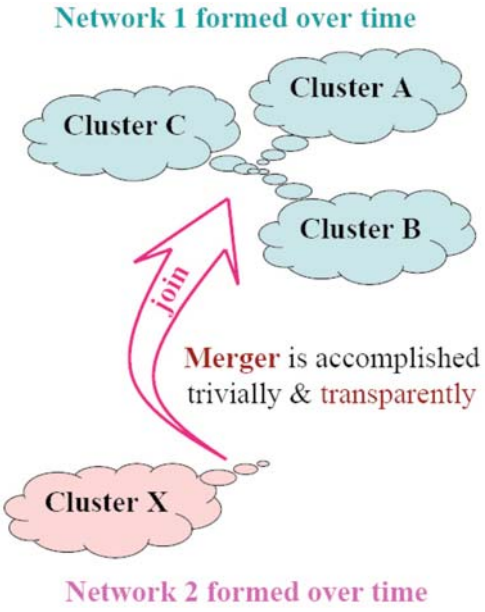
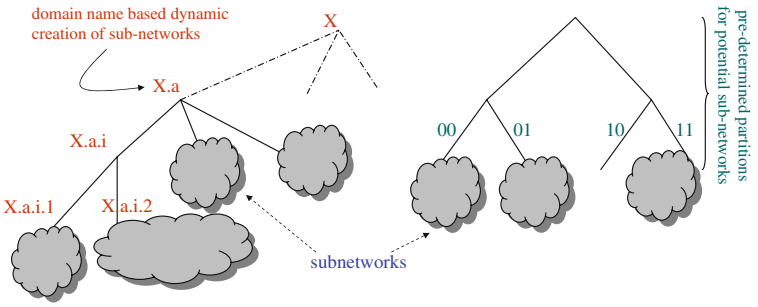



**Fig. 9** Merger of two unstructured overlay networks (like Gnutella) is trivial. As soon as some peers from each of the two originally isolated networks establish connections to each other, a merged network is formed

networks act as "sub-networks" which are glued together to form one integrated network. Two addressing components are used to uniquely determine a peer's role in the integrated network – one to determine which autonomous sub-network it belongs to, and the other to identify it within the sub-network. The sub-networks can be distinguished based on a domain name space [13] or by simply allocating a predetermined part of the key space for each potential sub-network [6]. Figure 10 shows example instances of such hierarchically integrated overlays. Such hierarchy allows nodes to retain communication traffic within a particular domain, or look for keys available within specific sub-networks instead of the whole integrated network. On the downside, each peer needs to keep track of more information, including keeping track of which sub-network it belongs to, at different levels of the hierarchy. For example, in a hierarchical Chord using the Canon approach each peer also needs to maintain a list of successors (and hence a ring) at every level of the hierarchy. Thus the simplicity of a completely flat address space which was a design goal in many original distributed hash tables is lost.




The `Canon'approach where sub-networks are distinguished by domain names. A parent domain emulates a merged network of the children sub-networks. Each individual subnetworkcan, in theory, use its own routing mechanism.

The `Cyclone'approach where inter sub-network routing is based on XOR distance between subnetworkidentifiers, while intra-subnetwork traffic uses proprietary routing mechanism of the subnetwork.

**Fig. 10** Hierarchically integrated networks based on Canon [13] and Cyclone [6] approaches

An alternative to the hierarchical approaches discussed above is to device mechanisms to merge the originally isolated networks, so that a single resulting network is formed. That way peers do not need to keep track of the hierarchy, and the original overlay (DHT) design of completely flat identifier space is achieved. Some of the challenges of merging similar structured overlays, which is essential for decentralized bootstrapping of overlays [11], has been studied by Datta et al. [10], which we summarize next along with tentative solutions and shortcomings. The discussion below is also pertinent, at least in part, to hierarchical approach like Canon [13], where the authors overlook the key management issues.

## *5.1 Merger of Two Ring Based Networks*

Consider two originally isolated Chord networks $\mathcal{N}_1$ and $\mathcal{N}_2$ with $N_1$ and $N_2$ peers respectively. An example of two such networks (superimposed) is shown in Fig. 11a. Next we will explain how such isolated networks can merge into one, since such an organic network growth is not only essential for decentralized bootstrapping, but also such a merger is necessary to ensure that the overlays retain functionality. When peers from the two different overlays meet each other (by whatsoever reason – accidentally or deliberately), in a decentralized setting there is no way for them to ascertain that they belong to two completely different systems. This is so because overlay construction always relies on such peer meetings to start with. As a consequence, if the peer pair that meets have identifiers such that they would replace their respective successor and predecessor, then they will indeed do that.

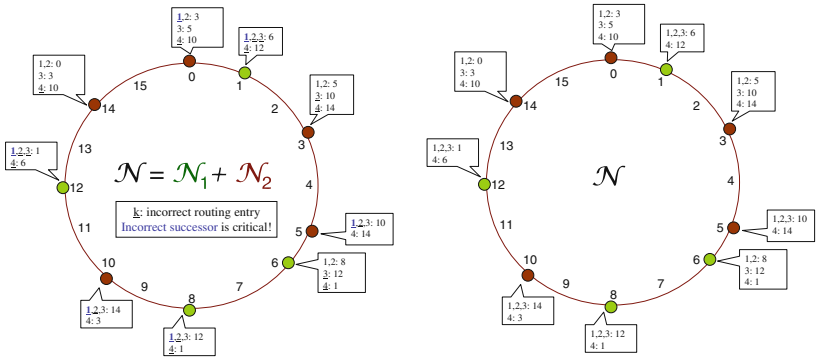



**Fig. 11** When (peers from) two ring-based overlays meet

For our example from Fig. 11a lets say peer 1 from $\mathcal{N}_1$ meets peer 0 from $\mathcal{N}_2$. Then peer 1 will treat 0 as its new predecessor, and 0 will treat 1 as its new successor, instead of 12 and 3 respectively. However, if they only change their local information, then the ring network will no more be strongly stable (may in-fact not even be weakly stable). In-fact such a reconfiguration will need and lead to a cascading effect, so that all members of both the original network try to discover the appropriate immediate neighbors (successor/predecessor) – requiring coordination among all the peers.

*Estimation of the probability that a peer's predecessor changes:*

From the perspective of any peer in $\mathcal{N}_1$, the successor will change, if at least 1 out of the $N_2$ peers have identifier within the next $1/N_1$ stretch of the key-space (for which its present successor is responsible, on an average). Any particular peer from $\mathcal{N}_2$ has an identifier for this stretch with probability $1/N_1$. The number of peers falling in

this stretch is thus distributed as $Binomial(N_2, 1/N_1)$, which approaches to a Poisson distribution with expectation $N_2/N_1$ as $N_2 \to \infty$. Hence, a peer from $\mathcal{N}_1$ will have its successor unchanged with probability $e^{-\lambda_1}$ where $\lambda_1 = N_2/N_1$. Thus each of the $N_1$ peers will have their successor node changed with a probability $1 - e^{-\lambda_1}$ i.i.d. Peers in $\mathcal{N}_2$ will be affected similarly with a parameter $\lambda_2 = N_1/N_2$ (symmetry).

*Estimate of the number of peer pairs which will have their immediate neighbors (either successor and/or predecessor) changed:*

The number of peers whose successor will change in $\mathcal{N}_i$ is then distributed binomially $Binomial(N_i, 1 - e^{-\lambda_i})$ for i =1,2. Hence the expected number of nodes which will need to correct their successor nodes (and predecessor nodes) is $N_1(1 - e^{-\lambda_1}) + N_2(1 - e^{-\lambda_2})$.

The basic idea of how the ring can be reestablished is that when two peers from different networks meet so that they replace each other's successor and predecessor (immediate neighbor), then this information needs to be communicated to the original immediate neighbors, and the process continues.

There are several combinations of how the neighborhoods of the peers are affected after their interaction, each of which needs to be accounted for the actual network merger algorithm. Moreover, different combinations of faults (single or multiple peers crashing or leaving) can happen during the ring merger, and these too need to be dealt with. The specifics of such algorithms, and evaluation of the actual ring network merger algorithm is currently underway.

Without proper and exhaustive evaluation of the exact algorithms for merging two ring networks, it is difficult to see whether a strongly stable ring can be directly achieved, or whether a sequence of faults during the merger of two rings can even lead to a loopy network, which would then require even more effort to converge to a strongly stable state using Chord's already existing self-stabilizing mechanisms.

Thus the back of the envelope analysis above just provides the expected lower-bound of the ring reestablishment process in terms of correction of successor/predecessors. The latency of such a process started because of two peers from the two networks will be $O(N_1 + N_2)$ even if there is no membership changes during the whole merger process – this is the time required to percolate the information that the ring neighborhood has changed and to discover the correct neighbor when peers from both the original networks are considered together.

### 5.1.1 Ring Loses Bearing During the Merge Process

Above we provided a sketch of how to only reestablish the ring topology – which only guarantees the functional correctness of the routing process – i.e., the query will be routed to the peer which is supposed to be responsible for the key-space to which the queried key belongs.

Reestablishing the ring will be necessary in order to be able to query and locate even the objects which were accessible in the original network of any individual network. Hence, such a merger operation of ring topology based overlay will typically cause a complete interruption of the overlay's functioning.[5]

### 5.1.2 Managing Keys on the Merged Ring

Establishing the ring in itself is however not sufficient in an overlay network based index. In order to really find all keys (which originally existed in at least one of the two networks) from any peer in the merged network, it will still be necessary to transfer the corresponding key/value data to the "possibly" different peer which has become responsible for the new overlay. To make things worse, in a ring based network the queries will be routed to the new peer which is responsible for a key, so that even after the reestablishment of the ring itself, some keys that could be found in the original networks may also not be immediately accessible, and will need to wait until the keys are moved to the new corresponding peer.

Lets consider that before the networks started merging, network $\mathcal{N}_i$ had key set $\mathcal{D}_i$ such that $|\mathcal{D}_i| = D_i$. Furthermore, if we consider that $\alpha$ fraction of the keys in the two networks is exclusive, that is $|\mathcal{D}_1 \cap \mathcal{D}_2| = \alpha|\mathcal{D}_1 \cup \mathcal{D}_2|$, then on an average, if a $\mathcal{N}_i$ node's successor changes, it will be necessary to transfer on an average $\alpha$ fraction of the data from network $\mathcal{N}_j$'s $\frac{1}{N_1+N_2}$ stretch of the key-space. Thus, on an average, the minimum[6] required transfer of unique data from members of original networks $\mathcal{N}_j$ to $\mathcal{N}_i$ will be $D_{tx}^{j\rightarrow i} = N_1(1-e^{-\lambda_1})\alpha\frac{D_2}{N_1+N_2}$.

Apart from assigning the data corresponding to a key on the key-space to the peer which is the successor for that key, ring based topologies provide fault-tolerance by replicating the same data at $f_s$ consecutive peers on the ring.[7] Given the strict choice of $f_s$ as neighborhood changes, the transferred data will in-fact have to be replicated at the precise $f_s$ consecutive peers of the merged network, determining the actual minimal bandwidth consumption. Similarly, some of the original $f_s$ replicas will need to discard the originally replicated content.

---

[5] Note that such a vulnerability may expose ring topologies to a new kind of "throwing rings into the ring" distributed denial of service (DDoS) attack, though the implications of such an attack and the amount of resources an adversary will require to make such a DDoS attack needs to be studied in greater detail.

[6] The actual implementation of such a data transfer will need to identify the distinct data in the two networks and transfer only the non-intersecting one, in order to achieve this minimal effort. This is an orthogonal but important practical issue that any implementation will need to look into.

[7] The parameter $f_s$ is a predetermined global constant determined by the system designer.

### 5.2 Merger of Two Structurally Replicated P-Grid Networks

In the P-Grid network, replication is achieved in a very different manner than in a ring based overlay. Multiple peers are responsible for (by replicating) precisely the same exclusive key-space partition. This is called structural replication, and is in contrast to the ring based approach, where the replication is done along something like a sliding window, at the next $f_s$ peers on the ring. Effectively structural replication can be thought of as multiple virtual instances of the same key-space partitioned network superimposed with each other. For example, in Fig. 12a the network $\mathcal{N}_2$ can be thought of composed itself of superimposition of two networks, the first say comprising of nodes $U$, $W$ and $X$, while the other nodes belonging to the other virtual network. The routing reference maintained by the peers are intertwined, and does not need to discern the virtual networks, and is indeed desirable to achieve greater interconnection of the routing network.

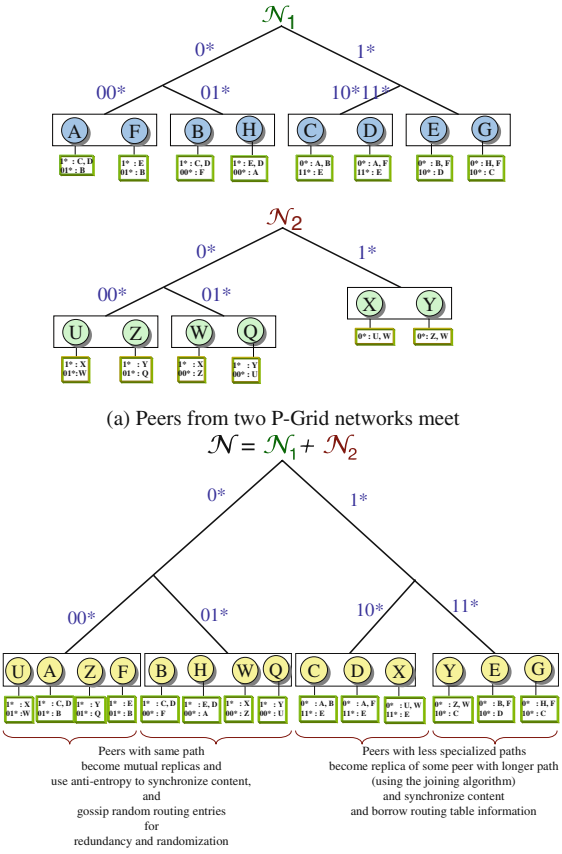
When peers from different overlays meet, similar to the situation in the ring networks, it will be desirable to achieve a merger of the two into a single intertwined network. Figure 12 shows an example of such originally isolated overlays, and the subsequent single merged overlay.

Notice that the originally isolated overlays $\mathcal{N}_1$ and $\mathcal{N}_2$ do not have identical partitioning of the key-spaces, but the eventual merged overlay shown in Fig. 12b should. Next we sketch how such a merger process will happen.

If peers from the two different networks meet, so that their paths are exactly the same (for example peers $A$ and $U$ from networks $\mathcal{N}_1$ and $\mathcal{N}_2$ respectively in Fig. 12a), then they will execute an anti-entropy algorithm to reconcile their content and become mutual structural replicas. In fact, such an anti-entropy algorithm will have to be run among all the other structural replicas of that part of the key-space too, and eventually of the other parts as well. However, since the original members of each network still retain the original routing links, routing functionality is not affected – and whichever keys were originally accessible will continue to be accessible. So to say, peer $C$ will always be able to access all the keys/content available at $A$ before the merger process. The keys from the same key-space which were present in the other network would however be available only after the background replication synchronization has completed. That is to say, a resource available originally only in $\mathcal{N}_2$ at $U$ and $Z$ (but with the same prefix 00 as $A$) will be visible to $C$ only when $A$ has synchronized its content with any one of $U$ or $Z$.

Use of structural replication has an additional downside – by not limiting the number of replicas nor having a proper structure among the replicas, it is difficult to have knowledge of the full replica subnetwork at each peer, and hence updates and replica synchronization is typically probabilistic. In contrast, once the ring is reestablished, replica positions are deterministic and hence locating replica is trivial in ring based topologies. Having discovered a replica, the anti-entropy algorithm itself (is an orthogonal issue) and hence the cost of synchronization of a pair of peers will be the same.

When two peers from $\mathcal{N}_1$ and $\mathcal{N}_2$ meet so that one's path is strictly a prefix of the other peer' path, then the peer with shorter path can execute a normal network




(a) Peers from two P-Grid networks meet

(b) Ideal P-Grid network comprising peers from both networks

**Fig. 12** When (peers from) two structurally replicated overlays meet

joining algorithm [2] – extending its path to replicate either the peer it met, or a peer this peer refers it to. For example, $Y$ may extend its path from 1 to 11. In order to do so, $Y$ will need to synchronize its content with one of the peers which originally had the path 11, say $G$. Moreover $Y$ will need to obtain routing reference to a peer responsible for the path 10 (e.g., peer $C$) – information it can obtain from $G$ itself.

Since new peers join as structural replica or existing peers, no other existing peer need necessarily to update their routing table for routing functionality (unlike in a ring based topology). Thus, peer $Q$ referring to $Y$ for prefix 1 continues to refer to it as such, and any query 10 from $Q$ is routed first to $Y$, which then forwards it to – say $C$. Peers may however, over time add more routing entries, for instance, $Q$ adding a reference to $D$ for redundancy in its routing table for the prefix 1. Such changes however is a normal process in the P-Grid network and can be carried on in the background, again without interrupting the functioning of the overlay (and in fact instead making it more resilient to faults and churn).

Consequently, neither joining peers, nor merger of two existing overlay network disrupt the available functionality of the network members.[8]

If peers with different paths meet each other, they need to do nothing, though they can refer each other to peers which are most likely to have the same path (similar to ring based topologies which can forward the peers closer to their respective key-spaces).

### 5.2.1 Managing Keys in the Merged Network

The amount of data that needs to be transferred from each system to the other is essentially the non-intersecting data. However, there is no need to transfer data from one peer to another merely because the key-space partition a peer is responsible for changes – because with structural replication, new peer joins or network mergers do not in itself automatically change the network's structure.[9]

The important thing to reemphasize is that a peer always finds the keys it could find before the merger process began, irrespective of the state of the merger process. Hence the replica synchronization can be done as a slow back-ground process – hence the performance and network usage is also graceful – that is, merger of two overlays does not suddenly overburden the physical network's resources nor disrupt the functioning of the overlay networks. Such a graceful merger of existing networks also facilitates highly parallelized overlay construction [2] in contrast to the traditional sequential overlay construction approaches.

## 6 Summary and Conclusion

Research and development in structured overlays now spanning almost a decade, has focused on diverse issues. Adaptation of different topologies in a peer-to-peer environment characterized by large scale, peer autonomy and membership dynamics, maintenance of these topologies – both in terms of ensuring that the topology invariants such as the ring invariant are continuously satisfied, as well as that other performance related concerns like load-balancing are addressed. These activities account for the first five-six years o structured overlay research, bringing it from the drawing board of theoretical results and simulation based validations starting around 2000–2001 to the actual prototyping and benchmark experiments in moderate scale in a controlled environment and with adequate coordination around 2005–2006. The final essential ingredient for a large scale deployment of structured overlays is to allow merger of smaller overlays to form a larger one organically. These smaller over-

[8] Note that the above discussion is true only for write once and then onwards read-only data, since for read/write, it will be necessary to maintain the replicas more pro-actively.

[9] Local view of the structure however changes when a peer with shorter path meets a peer with longer path, and extends its own path according to the network construction algorithm [2], as explained above.

lays can be bootstrapped using any of all the possible manners – quasi-sequential, parallelized, or by merger of even smaller networks. This chapter provides a high level summary and survey of the various kinds of bootstrapping mechanisms for some of the predominant classes of structured overlays.

## Acknowledgement

The work presented in this article is partially supported by A*Star SERC Grant No. 0721340055.

Disclaimer: This article summarizes different bootstrapping mechanisms for structured overlay networks, including approaches designed by third parties as well as myself along with various collaborators. The different approaches have accordingly been cited so that each mechanism can be attributed to their original designers.

## References

1. K. Aberer (Conference on Cooperative Information Systems (CoopIS 2001)) P-Grid: A self-organizing access structure for P2P information systems.
2. K. Aberer, A. Datta, M. Hauswirth and R. Schmidt (VLDB 2005) Indexing data-oriented overlay networks.
3. K. Aberer, A. Datta and M. Hauswirth (Self-* Properties in Complex Information Systems, "Hot Topics" series, LNCS, 2005) Multifaceted Simultaneous Load Balancing in DHT-based P2P systems: A new game with old balls and bins.
4. S. Abiteboul and I. Manolescu and N. Preda (SWDB 2004) Constructing and Querying Peer-to-Peer Warehouses of XML Resources.
5. D. Angluin, J. Aspnes, J. Chen, Y. Wu and Y. Yin (SPAA 2005) Fast construction of overlay networks.
6. M.S. Artigas, P.G. Lopez, J.P. Ahullo and A.F. Gomez-Skarmeta Cyclone: A Novel Design Schema for Hierarchical DHTs, (P2P 2005).
7. J. Byers, J. Considine and M. Mitzenmacher (IPTPS 2003) Simple Load Balancing for Distributed Hash Tables.
8. M. Castro and P. Druschel and A-M Kermarrec and A. Rowstron (ACM SIGOPS European Workshop 2002) One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks.
9. I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, B. Wiley (IEEE Internet Computing, vol.6 no.1, 2002) Protecting Free Expression Online with Freenet.
10. A. Datta and K. Aberer (IWSOS 2006) The challenges of merging two similar structured overlays: A tale of two networks.
11. A. Datta (SASO 2007) Merging Intra-Planetary Index Structures: Decentralized Bootstrapping of Overlays.
12. A. Datta EPFL Phd. Thesis 3615 (2006) SoS: Self-organizing Substrates.
13. P. Ganesan, K. Gummadi and H. Garcia-Molina (ICDCS 2004) Canon in G Major: Designing DHTs with Hierarchical Structure.
14. S. Girdzijauskas, A. Datta and K. Aberer (International Workshop on Networking Meets Databases, NetDB 2005) On Small-World Graphs in Non-uniformly Distributed Key Spaces.

15. S. Girdzijauskas, W. Galuba, V. Darlagiannis, A. Datta and K. Aberer (accepted for publication in Springer's Peer-to-Peer Networking and Applications Journal) Fuzzynet: Zero-maintenance Ringless Overlay.
16. A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu and I. Tatarinov (TKDE vol.16 no.7, 2004) The Piazza Peer Data Management System.
17. IETF-RFC:3174 (http://www.ietf.org/rfc/rfc3174.txt, 2001) Secure Hash Algorithm 1 (SHA1).
18. M. Jelasity and O. Babaoglu (ESOA 2005) T-Man: Gossip-based overlay topology management.
19. M. Jelasity, A. Montresor and O. Babaoglu (IEEE International Conference on Distributed Computing Systems Workshops, 2006) The Bootstrapping Service.
20. J. Kleinberg (STOC 2000) The Small-World Phenomenon: An Algorithmic Perspective.
21. G. Koloniari and E. Pitoura (EDBT 2004) Content-Based Routing of Path Queries in Peer-to-Peer Systems.
22. P. Maymounkov and D. Mazieres (IPTPS 2002) Kademlia: A peer-to-peer information system based on the XOR metric.
23. A. Montresor, M. Jelasity and O. Babaoglu (P2P 2005) Chord on Demand.
24. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst and A. Löser (Journal of Semantic Web, vol.1 no.2, 2004) Super-peer-based routing strategies for RDF-based peer-to-peer networks.
25. C. G. Plaxton, R. Rajaraman and A. W. Richa (SPAA 1997) Accessing Nearby Copies of Replicated Objects in a Distributed Environment.
26. A. Rowstron and P. Druschel (Middleware 2001) Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.
27. I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan (SIGCOMM 2001) Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications.
28. B.Y. Zhao, J.D. Kubiatowicz and A. D. Joseph (2001 UC Berkeley Technical Report UCB/CSD-01-1141) Tapestry: An infrastructure for fault-tolerant wide-are location and routing.