

# Merging Intra-Planetary Index Structures: Decentralized Bootstrapping of Overlays

Anwitaman Datta

E-mail: anwitaman@ntu.edu.sg

School of Computer Engineering, NTU Singapore.

## Abstract

*Peer-to-Peer index structures distributed and managed over the planet, commonly known as structured overlays (e.g., Distributed Hash Tables), are posed to play the role of a fundamental building block for internet-scale distributed applications and information systems. One of the biggest impediment in realizing index distributed at intra-planetary scales is to be able to merge distinct indices that might have been constructed independently and separately, or have resulted from network partitions. By facilitating merger of such isolated index-structures, decentralized bootstrapping of structured overlays is made possible. We argue that such a self-organizational attribute of decentralized bootstrapping is of paramount importance for large scale systems. In this paper we provide algorithms and simulation based evaluation of merger of two tree-structured overlay networks. The experiments validate some intuitions, particularly we see how the merger operation can be carried out transparently from the end users in the sense that all keys that are accessible to any peer prior to the merger process continue to be accessible during and after the merger. Surprising at a first glance, we also observe that the overhead of data movement during merger of two overlays which originally partition the key-spaces arbitrarily is higher if there is more common content stored in the two networks. This counterintuitive behavior is on account of the fact that the assignment of partitions to peers change during the merger process, which in turn leads to movement of data. Since peers act based on local knowledge, the data already present in the other network is still unnecessarily moved, adding to the data transfer overhead.*

**Keywords:** Peer-to-Peer, Structured Overlay, Decentralized Bootstrapping, Overlay Merger, Scalability.

## 1 Introduction

The recent years have witnessed increased use of resources at the edge of the network - typically desktop com-

puters interconnected across the internet provide services and run applications in a peer-to-peer manner, as an alternative to the traditional paradigm of using dedicated infrastructure and centralized coordination and control. Such overlay network technologies have found diverse applications spanning networking, information systems and distributed systems domains. Overlays are used for supporting various communication primitives like multicast, QoS and more reliable routing which are not readily supported by the internet infrastructure. From information systems perspective, the internet along with its end users is transmogrifying into a planetary scale file system and repository for information and content. P2P data management and information systems need index-structures. Structured overlays - distributed hash tables [13, 14] as well as range partitioned [1, 5] are such distributed index structures.

The operational phase of such structured overlays is often decentralized (including self-maintenance). Construction or bootstrapping of overlays has traditionally assumed a quasi-sequential approach, in which new peers would join by contacting some existing members of the network [5, 13, 14]. More recently, parallelized structured overlay construction mechanisms have been developed [1, 11], again assuming participating peers already know and can contact each other, e.g., using an unstructured connected network. All these structured overlay bootstrapping strategies thus implicitly assume logical centralization - even if distributed, e.g., a set of globally known rendezvous "seed" peers - which facilitates new peers to join and form one single network.

Imagine that any or several of these existing techniques are used to construct individual overlays based on distinct (disjoint) set of *seed* peers. These overlays will operate as isolated islands. If at a later time, some peers from one overlay network meets peers from another network (by whichever mechanism), existing overlay maintenance and construction mechanisms will fail to deal with it [4]. Facilitating merger of such originally isolated but evolved overlays into a single overlay network paves the way for decentralized bootstrapping of overlay networks. People can

set up their own bootstrap seed peers to build smaller but functional overlay networks, and when they meet and wish to merge their network with others, the isolated overlays can coalesce to grow organically into a single larger overlay. Generally speaking, decentralized bootstrapping is a desirable self-organizational property for large-scale systems. Merger and thus decentralized bootstrapping occurs trivially in unstructured P2P systems, while current structured overlays lack it altogether.

In this paper we provide mechanisms for merger of structured overlays which is necessary for decentralized bootstrapping, for one major class of overlay networks which use prefix based routing and have tree topology, and evaluate the merger process based on simulations. Note that for multiple overlays to be able to coalesce, they should all be using the same protocol to start with. Inter-operability of multiple overlays, either based on same or different protocols is a related but distinct problem currently being researched in the networking community, and we will discuss and distinguish it from our objective in Section 2 on related works. We point out why such approaches diverge from some of the original goals of structured overlays and particularly are inadequate to address the need of data-oriented applications. For the sake of context and self-containment, a brief overview on structured overlays is provided in Section 3. In [4] we identified the distinct challenges faced by major classes of overlay topologies - tree and ring. In Section 4 we look at the challenges of merging overlays in general, and the algorithmic details and subtleties of merging tree structured overlays. Evaluation of our merger algorithms for a specific system - P-Grid [1] - is presented in Section 5. From the experiments we see that it is indeed possible to perform the merger process in a manner transparent to the end users, i.e., for any peer keys which were accessible to it prior to the commencement of the merger process continue to be always accessible to that peer. The experiments also reveal some apparently surprising behaviors, particularly pertaining to the volume of data transfer. In a typical setting the original networks are expected to have different key-space partitioning to begin with, and in this case, more common data in the networks at the beginning incurs more data movement overhead than if there were less data common in both networks. We conclude in Section 6. As noted in [4], merger of ring based overlays like Chord [14] is structurally a different problem. It is beyond the scope of this paper, and will be addressed separately. Furthermore, depending on application requirements, one may actually want to keep two distinct indices (and not merge them). When to merge or not merge, and how to expressly initiate such a merger is thus something to be decided by the application logic, and is not addressed here. The meeting of two peers from the originally distinct networks may happen by chance or design, and is again not

an issue we deal with here. We concern ourselves with only a mechanism to merge two such distinct overlays when necessary, given that at least one peer from one network gets to contact another peer from the other network.

## 2 Related work

The networking community has recently looked into similar problem of dealing with multiple overlays based on potentially diverse topologies and protocols. Since the objective has typically been of end-to-end communication of hosts, these approaches have borrowed ideas from the networking domain - managing individual overlays as ASs/domains, and routing traffic from one overlay to another using BGP like mechanisms based on certain peers specifically designated as gateways [9, 2]. In the discussion of [9] the authors themselves point out the limitation of their approach in dealing with application layer functionalities, e.g., storage. There are several other efforts which try to integrate multiple overlays based on the idea of domains [6, 7], and have the same limitations when it comes to support application layer functionalities.

In a flat (domainless) overlay, one needs to know only the name of the resource to be sought, and locate it. Such a data-structure can thus readily be used as an index. However if it is necessary to already know specific domain (which is okay for end-to-end communication), then it can not be used as an universal index as has been the case with flat domainless overlays. Such domainless flat data-structures was also one of the original appeals of universal applicability of first generation structured overlays like distributed hash tables (DHTs).

Association of content with peers can potentially change with merger of overlays. Communication oriented applications do not need to deal with management of the stored content, which is however a critical issue for any data-oriented application, particularly if full recall (=1) is desired or needed. This issue has been overlooked in previous works [2, 6] which otherwise propose ring merger. A gossip based bootstrapping service [8] has been proposed as a generic mechanism for constructing an overlay in a parallelized manner [11] from scratch. We speculate that similar gossip based approach can also be used to merge multiple overlays. If such an approach is however indeed used, again, in itself it will not deal with the key-to-peer associations, and thus will not be directly useful.

Merger of overlays shares some resemblance with another essentially different issue - churn (membership dynamics). Merger of two overlays, seen from the perspective of individual peers (which is how these decentralized systems operate) can look very similar to new peers joining the network.

However, churn is a gradual process, and the system

needs to continuously perform repair operations to rectify local view of peers in order to deal with the continuous membership changes. When two isolated networks need to merge, the magnitude of the population change with respect to their original population size is very high.

The problem we address in this paper, that of merger of two distinct overlays but using the same protocols is somewhat different from how peers across different overlay networks can establish end-to-end communication among themselves. For end-to-end communication, one can devise other strategies [9]. Merger of overlays and management of stored keys in such a merged network becomes necessary particularly when the overlay is used as a distributed index-structure and provide application layer functionalities, e.g. data storage [3, 10], which was one of the original purposes of such structured overlays.

### 3 Background: An overview on overlays

In recent years the concept of structured overlays has attracted a lot of attention because of its potential to become a generic substrate for internet scale applications.

Structured overlay networks comprise of the following three 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. A peer responsible for a particular key-space partition should have all the objects<sup>1</sup> which are mapped into keys belonging to the corresponding key-space partition.

(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 other partition - 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, and hence the values associated with the keys.

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. In storage and data-oriented applications, keys are generated using suitable (hashing) functions corresponding to the objects (values) to be stored or data to be indexed. The key-value pairs are then managed by the specific peers re-

<sup>1</sup>In this paper we use the words object, data or key-value pairs interchangeably. Additionally, when context allows, we refer to querying or accessing the key to mean the key-value pair. How an object is associated with a key is an orthogonal issue determined by application needs and does not directly concern the indexing layer.

sponsible for particular keys. Anyone who needs to access or manipulate the data thus needs to locate one of these responsible peers first. This is one of the principal functionality that structured overlays are designed to meet.

#### 3.1 Redundancy and resilience

A structured overlay network needs to meet two goals to be functionally correct in locating stored key-value pairs:

(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 (and ideally, none other).

We use the information retrieval metric of *recall* to quantify the overlay's functional correctness. It can be interpreted as the probability that a key-value pair can be located if the key-value pair is actually present in the network, by querying for the corresponding key at any peer in the network.

Redundancy is a time tested approach to provide fault tolerance and is widely used in structured overlays as well. Multiple routes from any peer to any target key-space partition provides static resilience as well as flexibility to choose alternative routes in order to deal with load-imbalance and congestion. Replication of each key at multiple peers guarantees availability of the keys despite the fact that individual peers may be unreliable, and is also useful to deal with higher load. Correctness of routing in a dynamic environment is achieved by maintaining the peers' routing tables correctly over time and using a proper routing algorithm. Correctness and completeness of binding is achieved by moving the corresponding keys (content) as and when the key space partition a particular peer is responsible for changes, and synchronizing the content among replica peers.

#### 3.2 Prefix-routing topology with structural replication (P-Grid)

In this paper we provide algorithms specifically designed to merge P-Grid [1] networks, which uses prefix based routing [12]. P-Grid abstracts a tree structure (Figure 1), and the algorithms we propose can be readily modified to merge other tree structured overlays.

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.

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

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. Figure 1(a) shows instances of P-Grid network (peer's path and routing table). e.g., in  $\mathcal{N}_1$ , peers  $A$  and  $F$  are mutual replicas and are responsible for the key-space with prefix 00. Peer  $A$ 's routing table comprise of peers  $C$  and  $D$  from the partition with prefix 1 and peer  $B$  from the partition with prefix 01. Note that while P-Grid abstracts a tree-structure, there is actually no hierarchy.

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.  $\langle \kappa(d), d \rangle$  is the key-value pair stored in the overlay.<sup>2</sup>  $\kappa(d)$  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*).  $\mathfrak{R}(\kappa)$  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

<sup>2</sup>Depending on implementation,  $d$  may actually be a (set of) pointer(s) to the actual content.

replicas  $\mathfrak{R}(\pi(p))$ , and use epidemic algorithms to maintain replica consistency. Routing in P-Grid is based on greedy prefix matching [12]. So a query at  $A$  for any key with prefix 11 will be forwarded to one of peers  $C$  or  $D$ , and then this peer would forward the query to  $E$ . Peer  $F$  would however have forwarded it directly to  $E$ .  $E$  is responsible for all keys with prefix 11, and should thus answer the query.

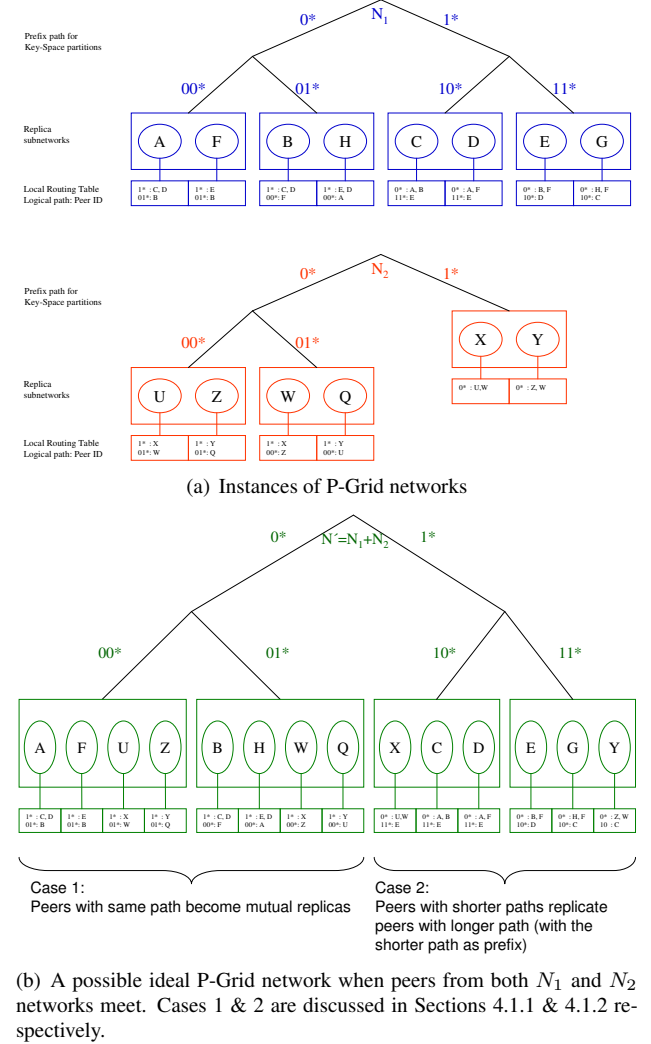


Figure 1. Merger of structurally replicated P-Grid networks

## 4 Merger operations

We denote peer  $p$  from network  $\mathcal{N}_i$  as  $p^{\mathcal{N}_i}$ . Furthermore, we consider only two originally isolated networks  $\mathcal{N}_1$  and  $\mathcal{N}_2$ . Extending our algorithms for merger of multiple networks is relatively straightforward subject to the ability to

distinguish networks (based on some network identifier). Management of such network identifiers in a decentralized fashion can however be a sufficiently involved problem in its own right.

#### 4.1 Meeting of peers from originally different networks $\mathcal{N}_i$ and $\mathcal{N}_j$

We consider the different actions necessary to merge two networks, once peers from two different networks meet. How the first such meeting occurs does not concern us. However in order to continue and complete the merger process, we induce further meeting of peers following this first (chance) interaction.

##### 4.1.1 Interaction of peers $p^{\mathcal{N}_i}$ and $q^{\mathcal{N}_j}$ s.t. $\pi(p^{\mathcal{N}_i}) = \pi(q^{\mathcal{N}_j})$

**Managing replica subnetwork (content synchronization):** 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 Figure 1(a)), then they will execute anti-entropy algorithm to reconcile their content and become mutual structural replicas.

Such anti-entropy algorithm is already used among replica peers to keep the content updated among themselves. So in our example, originally  $A$  and  $U$  upon interaction will synchronize their mutual content. Later,  $A$  may run anti-entropy with  $F$  (which it does regularly anyway), so that  $F$  will then obtain the data items which originated in network  $\mathcal{N}_2$ .

We need to ascertain that despite the network merger process, whichever keys were originally accessible will continue to be accessible. It is easily achieved. E.g., peer  $D$  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. So if  $D$  communicates with  $F$  after interaction between  $A$  and  $U$  but before an anti-entropy operation between  $A$  and  $F$ , then  $D$  may miss the keys which were originally available only in network  $\mathcal{N}_2$ . 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 definitely visible to  $D$  only when both  $A$  and  $F$  have synchronized their content, adding the key value pairs originally available only at  $U$  and  $Z$ .

Peers also update their view of the replica subnetwork for future runs of anti-entropy.

**Network rewiring (routing table updates):** One of the objectives of the merger process is also to update the rout-

ing tables at peers so that new routes among peers which belonged to the two originally different networks are established. The intuitive way to do it is to merge the peers' routing tables. So  $A$  can add an additional route to  $W$  (learnt from  $U$ ) for the prefix 01. If this is done, then routing a query from  $D$  for the prefix 01 may end up in either of  $B$ ,  $H$  or  $W$  since the actual route is chosen randomly at each hop out of all possible choices. This will however mean that keys which were originally deterministically accessible at  $D$  may not be anymore, and will depend on chance, until peers responsible for that portion of the key-space also complete the merger and synchronization process. Such effect is undesirable for most applications. The merger process should be transparent to end users, so that keys that were originally available should always continue to be accessible even during the merger process. Keys originating from the other network may then become available incrementally. It may be achieved variously. One possibility is to use all routing entries to forward queries at each level. This however leads to unnecessary traffic in the network. Furthermore, it will be difficult to change routing policy based on whether there is an ongoing merger process or not. So if this routing strategy is used, then there will be unnecessary traffic during the whole lifetime of the network, most of which is possibly when there are no ongoing merging process. An alternate strategy is to discriminate peers from originally different networks in the routing table references. So to say, for each level of the routing table, each peer may distinguish the peers which were originally in its network from the entries from the originally alien network that have been added during the merger process. Thus upon receiving a query, the peer forwards it to one appropriate peer which belonged originally to its own network, and one appropriate peer that it has learnt from the merging process. Moreover, this needs to be done for one query only once during the route forwarding process. Once the query is forwarded to a peer from the other network, it is in-fact desirable that it is circulated only among peers originally from that network in order to avoid redundancy and cycles.

In the long run, when the whole merger process is completed, i.e., when replicas have already run the anti-entropy algorithm among each other so that all peers from each subpartition have the same content, then the need to distinguish peers from different networks will cease to exist. However identifying completion of the merger process across the network is not tenable. A heuristic solution is to use a timeout  $\mathcal{T}_{out}$ . Thus, each peer locally decides to stop to distinguish peers from different networks after  $\mathcal{T}_{out}$  time since it last encountered a new peer from the other network executing the merger algorithm.

#### 4.1.2 Interaction of peers $p^{\mathcal{N}_i}$ and $q^{\mathcal{N}_j}$ s.t. $\pi(p^{\mathcal{N}_i}) \subset \pi(q^{\mathcal{N}_j})$

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's path ( $\pi(p^{\mathcal{N}_i}) \subset \pi(q^{\mathcal{N}_j})$ ), then the peer  $\pi(p^{\mathcal{N}_i})$  with shorter path can execute a normal network joining algorithm - extending its path to replicate either the peer it met, or a peer this peer refers it to based on other considerations like load-balancing. To keep things simple, here we chose the option where the peer  $p^{\mathcal{N}_i}$  simply decides to extend its path to replicate that of  $q^{\mathcal{N}_j}$ .

For example,  $Y$  may extend its path from 1 to 11 upon an interaction with either of  $E$  or  $G$ .

Once the peer extends its path, some of the keys stored at the peer ceases to be associated with the refined partition. For example, when  $Y$  extends its path from 1 to 11, as per the overlay's key-space partitioning principles, it is no more responsible for keys with prefix 10. These keys need to be shipped to other peers which are responsible for the corresponding prefixes. Pointers to suitable peers for these complementary subspaces where the key-value pairs need to be transferred can however be readily found from the routing table of the peer with originally longer path, i.e.,  $q^{\mathcal{N}_j}$ . In fact these entries are also used by the peer as its own routing table entries for the new extended path, since it has no preexisting pointers/routing table entries for the path extension. For example, since  $Y$  did not know any peer with prefix 10 originally, it uses the information obtained from  $G$ , and thus adds  $C$  in its routing table for resolving queries with prefix 10. Since there were no preexisting such routes locally, there is also no need to distinguish these newly acquired routing entries.

Anti-entropy based synchronization of content belonging to the partition (represented by  $\pi(q^{\mathcal{N}_j})$  and now also  $\pi(p^{\mathcal{N}_i})$ ) also needs to be done among the structural replicas. Similarly the view of the replica subnetwork needs to be updated. In contrast to the meeting of peers with same path, where the new replica subnetwork includes all peers from the two original subnetworks, in this case, original replicas of  $\pi(q^{\mathcal{N}_j})$  may decide to replicate a different part of the key-space, e.g.,  $X$  eventually replicates the partition with prefix 10.

The replica subnetwork view can be updated either proactively or lazily. We use a mixed strategy. Addition of new members in a subnetwork is done proactively. Omission of members from the subnetwork is done lazily during the anti-entropy process. So, when  $X$  eventually realizes that  $Y$  is no longer responsible for the same key-space partition (and vice-versa), the corresponding entry is dropped from the view of replica subnetwork. A peer which is not a replica is responsible for some other part of the network. Thus while the peer is dropped from the replica subnetwork view, the contact information can still be exploited by incorporating it as a redundant routing entry.

The data originally at a peer either resides at the same peer even after path extension, in which case, it continues to be accessible to all peers to which it was previously accessible. The data which however needs to be moved risks to become temporarily inaccessible. This risk is easily taken care of. The data is shipped to responsible peers following all available routes visible from the peer to which it originally belonged, i.e., using all its suitable new routing table entries. In our simplistic example, data with prefix 10 is shipped from  $Y$  to  $C$ , since  $Y$  has added  $C$  in its outgoing table. Thus, the data which was accessible to any peer (say  $Q$ ) originally because it had a route to  $Y$ , will continue to be available, since  $Y$  will provide necessary forwarding information.

This also means that because of the merger process, only peers directly involved in the specific interaction need to proactively update their local routing table. Routing table of other peers continue to be usable, and there is no cascading effect of routing table maintenance.

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 are normal in overlays and can be carried on in the background, without interrupting the functioning of the overlay and in fact instead making it more resilient to faults, churn and congestion.

Consequently, neither joining peers, nor merger of two existing overlay network disrupt the available functionality of the network members.<sup>3</sup>

Note that the merger process itself however needs to be performed in the whole network, and once the merger process starts, the first set of peers involved in the merger process can propagate the merger process rapidly within the rest of the network in a cascade, as will be discussed next.

#### 4.1.3 Cascading new interactions among peers from the originally different networks

The first meeting of peers from originally two distinct overlay networks may happen because of whatsoever reasons, with or without the explicit knowledge and intention of the interacting peers. Once such a first interaction occurs, if the interacting peers have the same path or one has a path that is strictly a prefix of the other, then they can execute the merger process as described above. If peers with different paths meet each other, they need to do nothing (in terms of merger operations), though they can refer each other using their routing entries to peers with longer mutual match of

<sup>3</sup>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.

path. Based on these references, each of the originally interacting peers will eventually discover peers from the other network, such that they have either the same path or one has a path which is strictly prefix of the other. Upon discovering such interaction partners, the peers can execute the merger operation as described above.

The merger operations are local. However, the same process needs to be executed over all the key space partitions. This is done by triggering new interaction among peers. Interacting peers can facilitate new interactions by informing all peers in their routing table about the new peer(s) they met. For example,  $A$  can inform  $B$ ,  $C$  and  $D$  about  $U$ . Each of them can then use the pointers available at  $U$  to identify suitable interaction partners, e.g.,  $W$  for  $B$ .

Spreading the information about the new network by flooding the local routing table is fast. All peers in the network gets informed in logarithmic time in terms of the network size (population). Moreover, many peers are subject to get multiple references. A final intuitive observation, also validated in simulations, is that a peer which has already executed the merger operation once does not need to initiate it again. However, if contacted by another peer from the originally different network, it may be required to perform again a merger operation.

## 5 Evaluation

The evaluations presented in this section are based on a discrete time simulator which models the peer interactions, routing table modification and data shipment. It also incorporates a modified version of the standard greedy prefix matching P-Grid routing algorithm to discriminate routing entries corresponding to peers from the originally different networks. Each of the experiments studied merger of two randomly generated P-Grid networks. Initially the networks were assumed to be in a consistent state. Particularly all replicas had consistent view of the stored content, and all routing entries at each peer was usable.

In real world P2P networks, these idealizing conditions do not hold due to continuous changes in the network membership (churn), and frequent addition of new content. These are important issues that have been extensively studied on their own right in the recent years. However in order to study only the effect of network merger, we conducted the current experiments ignoring these dynamics. Because of redundancy both in the routing entries as well as replication, it is expected that moderate level of churn will not affect the merger process, just like it does not affect normal operations in terms of recall (guarantee of success for a query) in the overlay, though it incurs additional maintenance overhead. This aspect is confessedly speculative and needs to be verified with further experiments. In that context, the current experiments are restricted to the primary

focus of this paper - algorithms for, and effect of, merger of two originally isolated overlays, ignoring other influences.

In each time step, some peers perform some of the following actions as applicable. Peers interact and perform necessary changes in the routing table, ship content (key-values) and disseminate and obtain information about potential interaction partners for next interaction. Between two time rounds, the state of the network just after a round is evaluated by processing queries. The queries are originated at randomly chosen peers, and the keys being queried are also randomly chosen. The performance is quantified in terms of success rate of the queries measured in terms of the information retrieval measure of *recall*.

**Performance metric:** We use *recall* as the primary performance metric.<sup>4</sup> So to say, if a particular key-value pair is stored in either of the two original networks, what is the probability of locating this key (and the corresponding value) starting from any arbitrary peer. Before the merger process, keys originally stored only in the other network can not be located. Keys present in the same network are originally accessible from any peer in the same overlay network. At the end of the merger process, it should be possible to locate any key starting the search at any peer using the overlay's routing algorithm. We see from the experiments that it is indeed the case. We use  $\mathcal{R}_{i/j}$  to denote the recall of keys originally stored in only network  $i$  (excluding common content), starting the query at a peer which originally belonged to network  $j$ .  $\mathcal{R}_{\cap/j}$  is used to denote the recall of content originally stored in both the networks, when the query is started at a peer from network  $j$ .

The cost of overlay merger operation is measured in terms of the volume of data transfer during these interactions and anti-entropy based synchronization. Of further interest is the latency of the merger operation.

**Parameter space explored in experiments:** We evaluate the process of overlay network merger in different settings. Particularly we have studied the effect on the performance of the network merger along three critical axis - size of networks  $|\mathcal{N}|$ , duplication  $\delta_{\mathcal{D}}$  of content in the original networks and the heuristic timeout  $\mathcal{T}_{out}$  parameter after which peers from originally different networks are not discriminated during routing. The duplication factor  $\delta_{\mathcal{D}}$  of content originally stored in the two networks is measured in terms of the fraction of keys stored originally in both the networks with respect to the ensemble of all keys stored in either of the two networks  $\delta_{\mathcal{D}} = \frac{|\mathcal{D}_1 \cap \mathcal{D}_2|}{|\mathcal{D}_1 \cup \mathcal{D}_2|}$  where  $\mathcal{D}_i$  is the collection of all data (key-value pairs) originally stored in network  $i$ .

<sup>4</sup>Precision is not an issue in the kind of search supported by structured overlays.

**Workload:** Network pairs of populations 17, 35; 132, 200; 723, 854 and 605, 612 were considered. All but in the last pair, the two networks had arbitrary and mutually different partitioning of the key-space. In the last case, both networks had arbitrary but exactly same key-space partitioning. For data duplication in the network  $\delta_D$ , values of 0.1, 0.2, 0.4, 0.5 were used. Sufficiently low value of  $\mathcal{T}_{out} = 3$  guaranteed perfect recall  $\mathcal{R}_{j/j}$ , and was used in most experiments. We also evaluate the behavior in the worst choice of  $\mathcal{T}_{out} = 0$ . In all the experiments, each key-space partition was originally replicated in between 3 to 5 peers. In each round, 2000 queries for keys present in at least one of the networks were issued, originating at random peers.

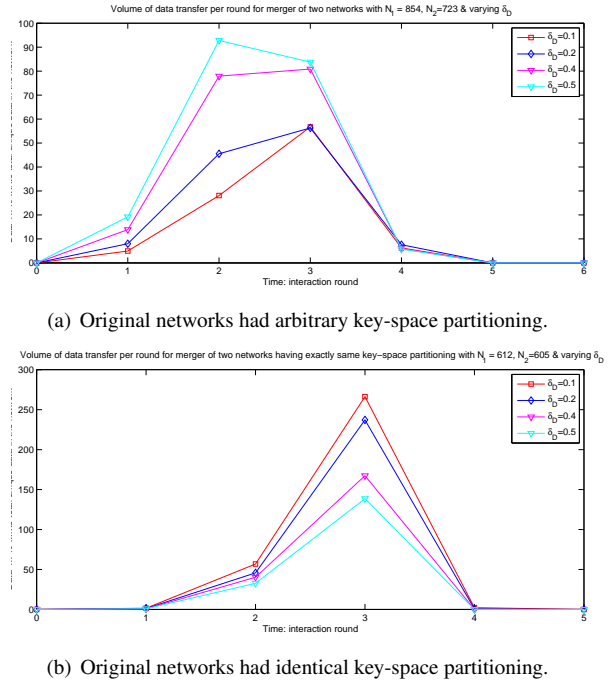
### 5.1 Recall and data transfer traffic

In the following plots, the  $x$ -axis represents the discrete time, and thus also indicates the rate and latency of the merger process. Recall of content originally present only in the other network  $\mathcal{R}_{j/i}$  is shown in Figures 2(a) and 2(c) for different network sizes. We see that  $\mathcal{R}_{j/i}$  rapidly climbs to 1, i.e., all keys even from the other network become accessible to all peers rather fast, and hence we can conclude that the merger process is also completed. All these experiments considered a value of  $\mathcal{T}_{out}$  such that the merger process is completely transparent to end users in terms of accessibility of content, particularly any key that was accessible to a specific peer continue to be accessible throughout the merger process, i.e.,  $\mathcal{R}_{j/j} = 1$ . Fairly low values of  $\mathcal{T}_{out}$  sufficed for the purpose, e.g., 3. We also noticed that even in the worst case in terms of choice of  $\mathcal{T}_{out}$ , the performance of the overlays in terms of accessibility of keys is marginally affected. We elaborate this subsequently in Section 5.2.

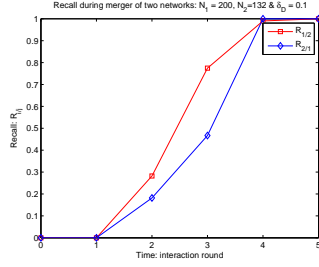
Volume of data transfer (normalized by the total number of unique data in both networks) is shown in Figures 2(b) and 2(d). Observe that in order to make all keys accessible to all peers, it is not needed that all keys which were not present in a particular network be shipped to peers in the other network. Instead often they become accessible simply because of the rewiring of the overlay. Thus the volume of data transfer (also accounting for the structural replica synchronization) is significantly lower than what would have been necessary if all data not present in the other network were to be transferred to some peers of the other network. Due to the infectious nature of triggering new interactions, most peers conduct the merger process rather fast, and the bulk of the data transfer is witnessed during these operations. Few peers actually need the background anti-entropy mechanism of replica subnetwork synchronization to obtain the keys from the other network.

As shown in Figure 3(a), for two networks with arbitrary and different partitioning of the key-space, the volume of data transfer increases if the two networks have originally

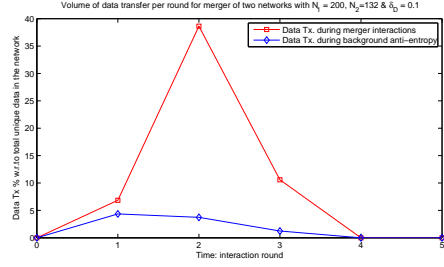
more content in common, i.e., high  $\delta_D$ . This is surprising since it may be expected that if more content is already present in both networks, it will need less data movement to make all keys available to all peers of both networks. Due to network merger the partition which individual peers are responsible for also changes, necessitating data movement as per the behavior described in Section 4.1.2. This data movement is actually a waste for duplicate content already present at the target peers, but is not known locally at the peers from which the data is being moved. This explanation of the counterintuitive behavior as observed in Figure 3(a) is also validated when we look at merger of two overlays with identical key-space partitions. In this case, there is no change of peers' partitions and thus no need to ship data away to other peers. The only data transfers in this case is because of anti-entropy based synchronization. We see that in this case the volume of data transferred decreases if the two networks originally had more content in common (Figure 3(b)). This observation is critical in that typically, two overlays are expected to originally partition the key-space arbitrarily, and hence differently. Thus in a typical situation, having less data in common in the original networks will make the merger process less expensive, in contrast to having more content in common, which will increase the data movement overheads of the merger process.



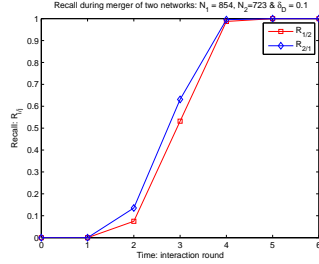
**Figure 3. Data transfer traffic for various values of  $\delta_D$ .**



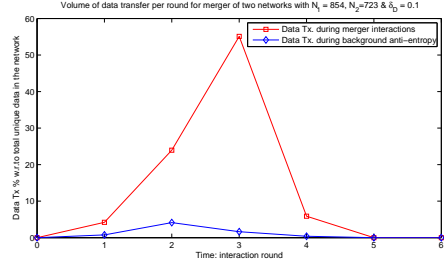
(a) Recall during the merger of two networks comprising 332 peers.



(b) Volume (normalized) of data transfer during the merger of two networks comprising 332 peers.



(c) Recall during the merger of two networks comprising 1577 peers.



(d) Volume (normalized) of data transfer during the merger of two networks comprising 1577 peers.

**Figure 2. Experiments based on merger of two P-Grid networks where  $\mathcal{T}_{out}$  was such that  $\mathcal{R}_{j/j}$  and  $\mathcal{R}_{\cap/j}$  were always 1 for  $j = 1, 2$ .**

## 5.2 Other scalability issues

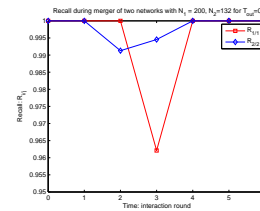
The latency of the merger process depends on the depth of the trees, and is essentially dictated by the smaller of the two networks involved in the merger. Route rewiring operations are also local.

Data transfer is bandwidth consuming, and is the greatest bottleneck for scalability. Moreover, its effect is not local among the interacting peers, and other peers are affected when a peer changes (extends) its path. However, it is an inevitable cost to incur if we want to merge two overlays. For practical purposes some optimizations are possible. For instance, if the data items themselves are rather large, one can reduce the actual traffic by sending first just the identifiers so that object already in the target peers is not transferred, as is happening with data items which were originally present in both networks. Since we measure the number of transfers, and not the absolute traffic, such implementation optimizations may have tremendous reduction of the absolute traffic, even though it does not change the fundamental findings of our experiments.

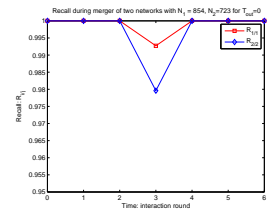
### 5.2.1 Effect of $\mathcal{T}_{out}$

$\mathcal{T}_{out} = 0$  is the worst case scenario for routing while the merger process is not complete since the peers from two networks are never discriminated during routing. Even then,

$\mathcal{R}_{j/j}$  is pretty high, though occasionally it is (slightly) less than 1 and this behavior holds for wide range of peer populations (Figure 4). Such a good worst case behavior in terms of choice of  $\mathcal{T}_{out}$  is reassuring, since even if a proper value of  $\mathcal{T}_{out}$  is not chosen, the performance deterioration is marginal. Since in large networks it is difficult to choose an appropriate  $\mathcal{T}_{out}$ , the merging networks' robustness against improper  $\mathcal{T}_{out}$  is desirable for the scalability of the merger process.  $\mathcal{R}_{\cap/j}$  was always 1 in all these experiments, thus content originally stored in both the networks always continued to be accessible.



(a) Merger of two networks comprising 332 peers.



(b) Merger of 2 networks comprising 1557 peers.

**Figure 4.  $\mathcal{R}_{j/j}$  in the worst case choice of parameter  $\mathcal{T}_{out} = 0$  (for  $j = 1, 2$ ).**

## 6 Discussion

This paper provides rigorous evaluation of merger of two structured overlays, in the process validating some intuitions and speculations [4] - particularly we see that it is indeed possible to perform the merger process in a manner transparent to the end users, i.e., for any peer, keys which were accessible to it prior to the commencement of the merger process continue to be accessible to that peer. The experiments also give some results which are surprising at a first glance, particularly pertaining to the volume of data transfer. In a typical setting the original networks are expected to have different key-space partitioning to begin with, and in this case, less common data in the networks at the beginning will incur less data movement overhead than if there were more data already common in both networks. We also see that the ad-hoc parameter ( $\mathcal{T}_{out}$ ) used in the algorithm for the merger/routing process have marginal effect, thus making the algorithm easy to use in real situations, without any need to determine an optimal parameter value. The current experiment results provide us a good insight into the merger process, however it also ignores the effect of churn in conjunction with the merger process, nor does it look into how multiple overlays may merge simultaneously. Since overlays should have sufficient redundancy to deal with churn in general, we expect that the merger process will stay rather unaffected by its influence. We assume peers locally discern between peers which are already part of its own network from peers from other networks, and use references from these alien peers to further propagate the merger process. Thus as long as we can ascertain a mechanism to distinguish normal churn (new node joining) from merger (meeting a node which is already part of another network), the merger mechanisms discussed in this paper are expected to work. These however remain speculations at the moment, and hence need to be verified. Besides more experiments to explore these aspects, theoretical results about the rate of convergence of the merger process, as well as overhead in terms of data movement estimated in terms of the initial conditions are required to completely understand the merger of overlays. Another aspect ignored in our current simulations is that the underlying physical network will have limited capacity, which may slow down the merger process since the merger process involves transfer of a huge volume of content. Furthermore, the routing on the overlays will be slowed in terms of real latency (not overlay hops), because of the high traffic and potential congestion caused due to data movement.

We argue that merger of structured overlays is a critical missing link, which facilitate decentralized bootstrapping and thus scalable deployment of such overlays in an organic fashion. In this paper we have a first concrete mechanism to merge a specific but important class of structured overlay

- tree topology (prefix based routing) - and rigorous if not exhaustive simulation based evaluation of the merger process in diverse settings. Further evaluations, development of a more concrete theory of the merger process, integration of the merger algorithms in the current P-Grid software and algorithms for merger of other topologies, particularly ring, are some of the crucial next steps.

## References

- [1] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. *31st International Conference on Very Large Databases (VLDB)*, 2005.
- [2] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on Flat Labels. In *SIGCOMM*, 2006.
- [3] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, 2001.
- [4] A. Datta and K. Aberer. The challenges of merging two similar structured overlays: A tale of two networks. In *International Workshop on Self-Organizing Systems (IWSOS)*, 2006.
- [5] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *VLDB*, 2004.
- [6] P. Ganesan, P. K. Gummadi, and H. Garcia-Molina. Canon in G Major: Designing DHTs with Hierarchical Structure. In *ICDCS*, 2004.
- [7] N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USITS 2003*, Seattle, WA, March 2003.
- [8] M. Jelasity, A. Montresor, and O. Babaoglu. The bootstrapping service. In *ICDCSW '06: Proceedings of the 26th IEEE International Conference Workshops on Distributed Computing Systems*, 2006.
- [9] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. OCALA: An Architecture for Supporting Legacy Applications over Overlays. In *USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.
- [10] J. Kubiawicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.
- [11] A. Montresor, M. Jelasity, and O. Babaoglu. Chord on Demand. In *IEEE P2P*, 2005.
- [12] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *SPAA*, 1997.
- [13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.
- [14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM, (Technical report version: <http://pdos.csail.mit.edu/chord/papers/>)*, 2001.