future internet

MDPI

Article

A Tale of Many Networks: Splitting and Merging of Chord-like Overlays in Partitioned Networks

Tobias Amft [1,†] and Kalman Graffi [2,*,†]

[1] Peopleware, Speditionstraße 5, 40221 Düsseldorf, Germany; tobias.amft@peopleware.com
[2] Faculty of Computer Science, Bingen Technical University of Applied Sciences, 55411 Bingen, Germany
* Correspondence: k.graffi@th-bingen.de
† These authors contributed equally to this work.

Abstract: Peer-to-peer overlays define an approach to operating data management platforms, which are robust against censorship attempts from countries or large enterprises. The robustness of such overlays is endangered in the presence of national Internet isolations, such as was the case in recent years during political revolutions. In this paper, we focus on splits and, with stronger emphasis, on the merging of ring-based overlays in the presence of network partitioning in the underlying Internet due to various reasons. We present a new merging algorithm named the Ring Reunion Algorithm and highlight a method for reducing the number of messages in both separated and united overlay states. The algorithm is parallelized for accelerated merging and is able to automatically detect overlay partitioning and start the corresponding merging processes. Through simulations, we evaluate the new Ring Reunion Algorithm in its simple and parallelized forms in comparison to a plain Chord algorithm, the Chord–Zip algorithm, and two versions of the Ring-Unification Algorithm. The evaluation shows that only our parallelized Ring Reunion Algorithm allows the merging of two, three, and more isolated overlay networks in parallel. Our approach quickly merges the overlays, even under churn, and stabilizes the node contacts in the overlay with small traffic overhead.

Keywords: peer-to-peer networks; dynamic overlay merging; distributed systems

check for updates

Academic Editor: Gianluigi Ferrari

Received: 24 April 2025
Revised: 17 May 2025
Accepted: 27 May 2025
Published: 31 May 2025

Citation: Amft, T.; Graffi, K. A Tale of Many Networks: Splitting and Merging of Chord-like Overlays in Partitioned Networks. *Future Internet* **2025**, *17*, 248. https://doi.org/10.3390/fi17060248

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Structured peer-to-peer (P2P) overlay networks are a class of distributed systems that enable scalable and decentralized data storage and retrieval. In contrast to unstructured overlays, structured overlays organize peers according to a deterministic topology, typically guided by a key-based routing mechanism [1]. In recent decades, different types of overlays emerged. According to the structure of the routing table and the usage of an identifier space, overlays can be categorized into various types like ring-based, tree-based, or mesh-based overlays. In this work, we focus on ring-based overlays, which are characterized by mapping nodes to points of a circular identifier space and by connecting peers according to their position in this identifier space. We choose Chord [2] as an example for this category of overlays, as it is the most cited structured peer-to-peer overlay. It allows detailed investigations on these issues, as solutions may be transferable to other ring-based overlay networks. Ring-based overlays such as Chord [2] or Pastry [3] assign node identifiers in a circular address space and connect nodes based on proximity in this space. This structure enables efficient lookup and robust self-organization, even under churn. One of the core strengths of P2P overlays is their resilience against central points of failure, making them a viable alternative to client–server systems, especially in environments with

unreliable or politically controlled infrastructure. Unlike centralized architectures, where content and metadata reside on a few dedicated servers, structured overlays distribute data and responsibilities across all participating peers, making surveillance and censorship significantly more difficult.

In contrast to the centralized server architecture, participants in a peer-to-peer network are expected to be unreliable and means for robustness are incorporated in the overlay protocols. User-related information is distributed among peers in a decentralized manner without the need for a dedicated server that could be attacked. The peer-to-peer architecture is, therefore, suitable for sharing data or information dissemination. An example of peer-to-peer-based social networking is LibreSocial [4], an online social network solely hosted by users. It uses the ring-based overlay network Pastry [3] and supports profiles, friend lists, and the storage of images and documents and enables real-time communication.

However, this robustness is challenged when large-scale network partitioning occurs, as seen during nationwide Internet shutdowns. Such partitions may fragment the overlay into multiple isolated subnetworks, impeding global consistency, searchability, and data availability. Once connectivity is re-established, these disjoint overlay instances must re-merge efficiently and without user intervention, a non-trivial task for most existing structured overlays.

Recent history offers several striking examples of network partitioning:

- During the Arab Spring in Egypt (2011), from 27 January 2011 until 2 February 2011, approximately 93% of Egyptian networks were unreachable (http://www.nytimes.com/2011/01/29/technology/internet/29cutoff.html, accessed on 26 May 2025), and most parts of the Internet in Egypt had been cut off by the Egyptian government (http://www.circleid.com/posts/egyptian_government_shuts_down_most_internet_and_cell_services/, accessed on 26 May 2025).
- Nepal cut off Internet access entirely in 2005, as did Myanmar two years later in 2007.
- A new report [5] lists further cases in Gabon (https://pulse.internetsociety.org/shutdowns/internet-shutdown-amidst-gabon-elections, accessed on 26 May 2025) (27 August and 14 September 2016, and starting from 26 to 30 August 2023), coinciding with the closure of polling stations after an election.
- In Iran, the Internet was shut down from 15 November to 24 November 2019 to "suppress the widespread aban-e-khoonin" movement. The report states that this was the most severe disconnection tracked by NetBlocks in terms of its technical complexity and scale (https://netblocks.org/reports/internet-disrupted-in-iran-amid-fuel-protests-in-multiple-cities-pA25L18b, accessed on 26 May 2025). In 2023, the initiative "Women, Life, Freedom" led to only a partial Internet shutdown.
- In Cuba, similar events were observed on 11 July 2021 (https://www.yucabyte.org/2023/01/25/cloudflare-radar-cuba-internet/, accessed on 26 May 2025) related to large protests against local COVID-19 regulations.
- Earthquakes, such as in Taiwan in December 2006 (http://news.bbc.co.uk/2/hi/asia-pacific/6211451.stm, accessed on 26 May 2025), might lead to Internet shutdowns due to link damages.
- A Shurfshark report on ongoing Internet limitations (https://surfshark.com/research/internet-censorship, accessed on 26 May 2025) shows that, in general, "protests were the primary trigger for Internet shutdowns", as [5] points out. A further useful tool to monitor ongoing shutdowns is listed by the Internet Society (https://pulse.internetsociety.org/shutdowns, accessed on 26 May 2025).

In cases where a network partitioning event occurs, participants of a disrupted peer-to-peer overlay should be capable of maintaining the network in corresponding geographical regions. The split of the overlay should survive, and the overlay network should quickly

stabilize. In addition, and more challenging, once the connectivity of the underlying network is re-established, various overlays should detect their presence and merge into one global overlay again.

To address this gap, we focus on the problem of merging ring-based structured overlays after a network partition. Our work is motivated by the need for resilient communication infrastructures in the face of technical failures, natural disasters, or political interventions.

The contributions of the paper are as follows:

- We elaborate on the problem of network partitioning in ring-based peer-to-peer overlay networks in Section 1.1 and discuss related work in Section 2.
- We present our distributed merging approach, named Ring Reunion, in Section 3; this Ring Reunion encompasses steps for identifying new overlays, coordinates the initialization of the merging process in the subnetwork, merges two disjoint rings from that elected starting point (in parallel), and terminates reliably.
- In Section 4, we evaluate our Ring Reunion approach in both simple mode and parallel mode (2^2–2^4 instances) in comparison to Chord–Zip [6] and two versions of Shafaat et al's Ring-Unification [7] approach (Simple and Gossip). We show that only our approach, Ring Reunion, is capable of merging the various Chord rings in demanding scenarios, such as merging multiple rings or having only very few contact points with other rings.
- Finally, we perform a parameter study with Ring Reunion and show the impact of parallelism (i.e., 2^2 to 2^6 parallel merging instances) and the impact of α, a parameter that specifies how many nodes in a subnetwork (of size $size_r$) should be in charge of initiating the merging.

Our approach for merging two or more ring-based overlays, Ring Reunion, is capable and efficient in merging disjoint rings after a network separation, such as that which occurs during an internet shutdown. Thus, local rings in countries can prevail during a local internet shutdown and are capable of rejoining after internet connectivity with other countries is reestablished. This general scenario can also be applied in other cases of network partitioning; thus, this is a concept that stabilizes ring-based peer-to-peer overlay networks.

1.1. Characteristics of Network Partitioning

As we know, autonomous systems (ASs) in the Internet are connected by Internet exchange points (IXPs) and controlled by different Internet service providers (ISPs). Although a geographical region holds no information about the topology of the Internet, we use the term region to refer to separated parts of the Internet. Using this term, we express the affiliation of an autonomous system, controlled by an ISP, to a country, a government, and other geographical or political regions. In this context, we denote the participants of an overlay which are able to communicate with each other after a network partition as part of a group.

While the separation of single nodes in peer-to-peer networks, i.e., churn, is a common challenge that researchers address regularly, the separation of whole groups is rarely considered. This happens when a set of nodes, which is associated with an autonomous system in the Internet, is separated from the existing overlay. This scenario could occur if all failing nodes are assigned to the same administrative domain which departs from the common network, e.g., if an autonomous system or a corresponding geographical region is suddenly disconnected from the Internet. Next, we discuss the two specific characteristics of overlay partitioning.

When two regions are isolated from each other and no routing between them is possible, it may be that more than two (Chord) rings will arise after a certain stabilize period. Although nodes of individual rings in the same region might be connected on network level, they could be incoherent in the overlay. Hence, during isolation, only those nodes are formed to a group, which are represented in the same set of finger entries and successor or predecessor pointers. Nevertheless, if the separated regions are reconnected again on network level, and both regions are capable of routing to each other, both regions will not converge to a common ring. The reason for this behavior is that no node in any separated overlay maintains any routing information about at least one contact node in the other separated overlay. Consequently, routing in a global overlay with both regions as participants is impossible. Both groups will remain independent from each other. In Figure 1, two connected regions are shown. In Figure 2, one region is isolated so that several rings are formed.

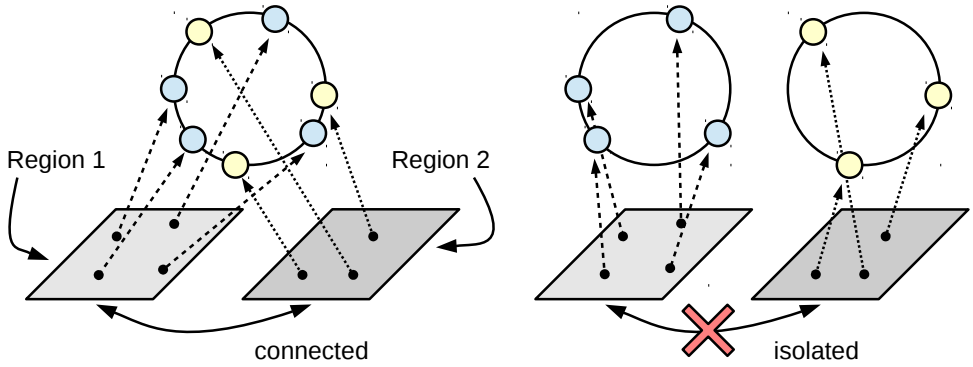


Figure 1. Ring composed of nodes from different regions is corrupted during isolation of the regions.

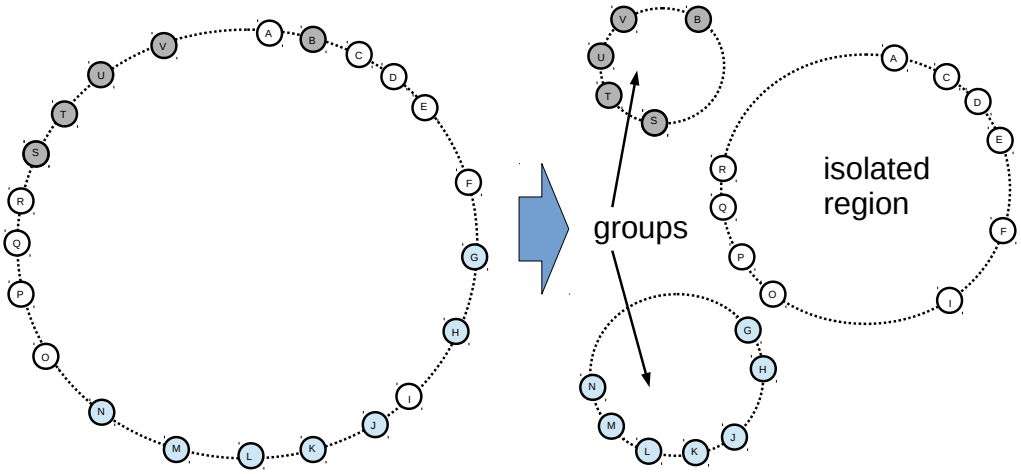


Figure 2. Several groups can form after one region has been isolated.

The functionality to merge other rings is not given in the original Chord and many related overlays. This functionality is highly desired as it would allow to use overlays in a broad range of scenarios, ranging from robust infrastructures to opportunistic networks.

1.2. Outline

In this paper, we investigate the question on how to merge separated ring-based overlays once the underlying networks are reconnected again. Any merger should operate independently, automatically, and without administrative restrictions. Specifically, we

investigate how to identify, from a single node's point of view, the existence of further overlay networks, how to initiate the merging, and how to terminate the merging process.

Therefore, we present in Section 2 a taxonomy of design decisions for merging algorithms and put related work in context. Motivated through this overview, we identify the weaknesses of current merging algorithms and present our proposal, the Ring Reunion Algorithm, in Section 3. In Section 4, we evaluate our proposed approach in comparison to Chord–Zip [6], the simple and Gossip-based Ring-Unification Algorithm [7], and the unmodified stabilization algorithms of Chord.

2. Related Work

Peer-to-peer-based overlay networks, i.e., structured overlay networks, provide key-based routing, as summarized in "Towards a Common API for Structured Peer-to-Peer Overlays" [1]. It is a common optimization goal to provide an efficient lookup interface through the overlay structure using a low hop count, few messages (i.e., low overhead), and aiming for a 100% lookup ratio. The network structure has demonstrated its scalability in various disruptive entries in den media industry [8], such as those shown by Skype, Spotify, Zattoo, and file sharing networks. Researchers in the literature discuss variants, how one can shape the overlay topology, how one can structure the routing table, and how one can deal with churn; these topics have been thoroughly investigated in peer-to-peer research. While research on peer-to-peer overlays has been popular in the last decade, the issue of maintaining such overlays in the presence of network partitioning has been discussed in only a few works.

Merging algorithms are used to stabilize overlays during network partitioning events by detecting and uniting participants that have become separated from the overlay. As fist step, a merging algorithm should be able to detect reachable nodes which have been separated. Nodes which belong to the same region especially should find each other quickly. Secondly, different groups which are connected in the underlying network should merge again. Once the connectivity of the underlying network is fully established again, the various overlay parts from different regions should detect their presence and merge to one global ring again. This niche of research question has not faced sufficient attention up to now. We summarize the available merging algorithms from the literature as we compare them to our Ring Reunion Algorithm in Section 4.

2.1. Re-Chord and Ca-Re-Chord—Self Stabilizing Chord Overlay Network

Re-Chord [9] is a version of Chord that has been extended through small modification to become self-stabilizing, i.e., the Chord network can recover from any initial state in which the network is still weakly connected. Using a single contact from another separated network might suffice for linearizing and merging the two networks. The paper shows that, in theory, within $O((log\ n)^2)$ communication rounds, the overlay is stable and reestablished again. While this characteristic has been proven theoretically, we simulated Re-Chord's behavior under churn in [10] and show that, in Re-Chord during the stabilization procedure, the lookup operation is drastically limited and objects cannot be found to a high degree, as long as nodes enter or leave any of the merging networks. Thus, we have proposed Ca-Re-Chord in [10] as an extended Re-Chord to address its struggles with churn and to stabilize the network despite nodes joining or leaving.

While Ca-Re-Chord is capable of handling simple churn, it cannot handle drastic events, such as instances when more than 50% of the nodes leave the network; this is a case which is assumed in a network separation. This paper directly aims to offer solutions for use in such drastic scenarios with the proposal of Ring Reunion; this can, in addition, also deal with more simple scenarios.

2.2. Datta et al.—Tale of Two Networks

In [11,12], Datta et al. characterize the challenges of merging two structured overlays which are based on the same protocol. The authors compare ring-based overlays (Chord [2]) with overlays which use prefix-based routing and structural replication (P-Grid [13]). In P-Grid, partitions and keys are represented as a set of *m*-bit identifiers, where m denotes the depth of a binary tree. Each peer corresponds to a leaf in a binary tree; consequently, each peer is associated with a path, which guides to a specific leaf. For the search and routing functionality, for each level of the binary tree, peers maintain references to other peers in complementary sub-trees. Since references to peers are chosen randomly; different instances of a P-Grid network may exist in parallel for a specific set of peers. The main difference of P-Grid to Chord and further popular overlays, such as Pastry, is that multiple peers are associated with exactly the same key space partition in order to improve fault tolerance. In other words, selected peers are mutual replicas which execute an algorithm to synchronize and update their content. Thus, the requirements for a valid state in Chord and Pastry are more stricter as peers in P-Grid take over the role of other peers before the state of the system turns from valid to invalid. We propose an approach that merges two or several sorted ring-based overlays into a single sorted ring-based overlay and enforces correct key responsibilities within a single key space.

2.3. Chord–Zip

The authors of Chord–Zip [6,14] suggest that the local information at two nodes from different Chord rings is sufficient to merge the two Chord rings. Each node is able to rearrange its locality without global knowledge. In cases where an initiator node becomes aware of any other contact node in another ring, it will be able to start a lookup to its successor in the other ring (denoted as alternative successor). Thereafter, the initiator node starts the merging procedure by forwarding its original routing information to the alternative successor and asks the alternative successor for its predecessor, its successor list, its finger table, and for the part of the identifier space it will be responsible for after the merger. Upon receiving the information from the initiator, the alternative successor responds by sending the requested information to the initiator. Simultaneously, the alternative successor forwards the merging process to the next contact node in the other ring, requesting successor list, predecessor, finger table, and range of responsibility. Obtaining all information from its neighbors, a node updates its routing links by using the most fitting entries in its routing table. The merging process is repeated by every node until the merger token arrives back at the initiator node. Finally, the initiator node combines its successor list and its finger table with those information it obtained from the alternative successor.

The execution time of the merger, which consists of one Chord join operation and the merge signal, can be improved by determining the multiple initiator instances that start the merger algorithm simultaneously. Further, initiator nodes block bypassing signals to terminate the algorithm.

For parallelization, the authors introduce two methods. First, nodes in both rings could start the merger algorithm concurrently. Secondly, some nodes could have the privilege of appointing further initiator nodes (e.g., finger entries).

Since the authors of Chord–Zip do not present lines of code for their algorithm, we interpret the Chord–Zip algorithm in the following way, as described in Figure 3. First, a lookup in the other ring is started to obtain the alternative successor. Thereafter, the merging algorithm is started by sending a ZIPPING message containing the initiator's routing information to the alternative successor. The receiver of a ZIPPING message answers with a ZIPPONG message containing its own routing information. The receiver of a ZIPPONG message on the other hand updates its routing table with the routing

information obtained from its neighbors. The initiator nodes stop the merging algorithm upon receiving a ZIPPING message. In the evaluation of this paper, we show that this Chord–Zip algorithm does not merge overlays reliably.

```
1  receipt of STARTZIPPER(contact) from m at n
2     sendto contact : LOOKUP (n.id)
3     if altSucc is received from contact within time interval γ then
4        sendto altSucc : ZIPPING (n.succ, n.succList, n.fingerTable)
5        n.isInitiator := true
6     end if
7  end event

8  receipt of ZIPPING (m.succ, m.succList, m.fingerTable) from m at n
9     sendto m : ZIPPONG (n.pred, n.succList, n.fingerTable)
10    if n.isInitiator = true then
11       n.combine()
12       n.isInitiator := false
13    else
14       sendto m.succ : ZIPPING (n.succ, n.succList, n.fingerTable)
15    end if
16 end event

17 receipt of ZIPPONG (m.pred, m.succList, m.fingerTable) from m at n
18    if n.isInitiator = false then
19       n.combine()
20    end if
21 end event
```

Figure 3. Chord–Zip Algorithm.

2.4. Shafaat et al.—Ring-Unification Algorithm (Simple and Gossip-Based)

In [7], Shafaat et al. present an algorithm for merging similar structured, unidirectional ring-based overlays. Their algorithm uses a parameter, termed fanout, to adjust the trade-off between message complexity and time complexity. Offline nodes in the routing table of a node are moved into a passive list of this specific node. Each node maintains an individual passive list. The nodes on this passive list are periodically probed. If one node is detected to be reachable again, a ring merging algorithm is started on both nodes. We present two versions of the Ring-Unification Algorithm in the following.

2.4.1. Simple Ring-Unification Algorithm

In the simple Ring-Unification Algorithm, each node uses a queue which contains any alive nodes detected in the passive list. In case it contains a node, this node, termed candidate, is considered as member of a potentially different overlay. Through the detection of aliveness, messages are exchanged and both the detecting and the candidate node are informed about each other. Both nodes start a lookup in their own overlay to the ID of the other node, thus identifying potential counterparts that are responsible for the same ID in the two different overlays. Once these counterparts are found, they are informed about the newly found node from the respective other region. Each node which learns about nodes from other regions determines whether the information about the newly found nodes fits better into the routing table. If so, a node updates its routing table and informs neighboring nodes about possible merging candidates. As a result, the merger proceeds in both directions, clockwise and anti-clockwise.

2.4.2. Gossip-Based Ring-Unification Algorithm

In addition to the simple Ring-Unification Algorithm, the Gossip-based Ring-Unification Algorithm starts multiple instances of the merger algorithm at random nodes, chosen with uniform distribution. The enhancement aims to increase the algorithm's performance during churn and other pathological scenarios that would immediately terminate

the simple Ring-Unification Algorithm. The fanout parameter is decreased every time a random node is picked, to ensure that a constant number of merger instances is created and to avoid an instance in which too many messages are produced. After evaluating the algorithm with different parameters, the authors suggest a fanout value of around 3–4 as a good trade-off between message overhead and time complexity.

2.5. Further Surveys on Overlay Network Management

Since 2015, only limited attention has been paid to the explicit merging of disjoint overlay networks—a challenge still faced by long-lived P2P systems subject to network partitions, e.g., through Internet shotdowns or extreme churn; while recent efforts have focused on ring optimization and fault tolerance (e.g., DGRO [15], PSPChord [16]), none address the reintegration of separate ring-based overlays—a gap we address with Ring Reunion. Further surveys on "Churn Handling Strategies to Support Dependable and Survivable Structured Overlay Networks" [17] and "Churn Handling Strategies for Structured Overlay Networks: A Survey" [18] elaborate on various stabilization strategies in (structured overlay networks from 2002 to 2020) but do not address the topic of completely partitioned overlay networks.

A third thorough overview on strategies against high churn is given in [19]; while strategies for increasing the overlay stability under churn are discussed, cases of network separation are only mentioned as a "current research challenge". A single paper is mentioned, namely [20], which looks into multi-dimensional routing inside partitioned social networks. The authors propose a localized focus of the network in order to support local communication in case of a partitioning. We proposed a similar concept (friend-to-friend routing) with FRoDO [21], Friendly Routing over Dunbar-based Overlays. Such focus on social networks limits the applicability of the overlay network, as it assumes a social/friendship relationship underlying the overlay network, which is not the case in regular structured overlays. In summary, even surveys on similar research question rarely address the issues of splitting and merging peer-to-peer overlays, or do not mention them at all.

2.6. Specific Overlay Types

Geography-based overlay networks, which aim to replicate the geographical structure of the nodes, are capable of modeling the local connectivity status of the nodes. Overlays such as Geodemlia [22], FRing [23], or GeoConnect [24] structure the network topology according to the geographical positions of the nodes. Thus, connectivity is mainly local (with some shortcuts to ensure logarithmic routing complexity). These networks are easier to merge, as the responsibility of various nodes is based on their location, and there is less competition in a partitioned network since no foreign node will claim geographical responsibility.

Network partitioning in decentralized systems has varying implications depending on the overlay structure and application semantics. In blockchain-based systems, partitions can lead to severe security risks such as chain splits, double spending, or consensus divergence. Since blockchain networks are primarily concerned with maintaining a consistent ledger across a dynamic and partially connected set of nodes, any disruption of peer connectivity may lead to forks or permanent inconsistencies in the global state. Several recent works have explored the robustness of blockchain networks under adversarial conditions. PeloPartition [25] considers the case of a global Internet outage and proposes a sharding mechanism to improve blockchain resilience, splitting the blockchain into branches with the assumption that they will be merged after the network recovers. Paphitis et al. demonstrate how a small number of targeted node removals can partition major networks like Bitcoin

and Ethereum [26]. In [27], the authors show that eclipse attacks can be used to isolate nodes and manipulate blockchain views in Ethereum. Saad et al. offer a broader analysis of how network partitioning affects both the security and performance of blockchain systems [28].

A core theme in blockchain-focused networks is the blockchain itself—that is, the data stored in the network. The connectivity of individual nodes is not of high importance, as all nodes eventually store the same data. This is different in regular overlay networks, where individual nodes are responsible for objects in corresponding ID ranges. Each node carries individual data that must not be lost. Thus, connecting each node to the correct network segment is essential—unlike in blockchain-based networks, where the redundancy of the blockchain mitigates such risks.

Peer-to-peer-based online social networks, as elaborated in a broad survey in [29], often reflect the social network ties in the overlay network topologies; this is the case in SafeBook [30], DiDuSonet [31], and Frodo [21]. The latter also shows that routing solely over a very small number of friends in a social network suffices in gaining complete connectivity. However, traffic in social networks is in contrast based on social network ties, which are different in regular ring-based structured overlays, where key-based lookups are quite evenly distributed in the ID space. Thus, p2p-based social networks might operate well in partitioned networks, as communication remains local, but ring-based overlays fail in large parts to resolve key-based lookups, as the keys and values are shut off.

2.7. Conclusions on Related Work

As only a few works in the literature address the research question of peer-to-peer overlay partitioning and merging, we highlight the few publications which do address it. The approach presented by Datta et al. in [11] assumes to have multiple responsible nodes for the same keys in the distributed hash table; while this is the case in P-Grid, for Chord and many other overlays, this is not the case. Chord–Zip and the simple Ring-Unification Algorithm on the other side have assumptions matching those of Chord, but only a single merger process traverses the overlay. We expect that, because this approach is both slow and sensitive to churn, the merger process might become lost and the merging might fail; this is an effect we also observed in the evaluation. The Gossip-based Ring-Unification Algorithm distributes the merging process based on the fanout parameter. However, as the node detection process is bound to the passive list, the set of potentially mergeable overlays is limited. Furthermore, here, we expect and show in the evaluation that the merging process can fail.

3. Our Merging Approach: Ring Reunion Algorithm

Next, we present our merging algorithm for ring-based peer-to-peer overlays, named Ring Reunion Algorithm. It aims on the one hand on reliable merging success in the presence of several parallel peer-to-peer overlays. On the other hand, the merging overhead in terms of bandwidth consumption and traffic should relate only to the number of overlay constructs (Section 4.1) in the network and not to the number of nodes. As constructs, we define patterns that form if nodes are interconnected by their successor pointers. We distinguish between three different constructs: full circles, hanger-ons, and chains (see Figure 4)—indicating whether the nodes are connected to a ring, a chain that is attached to other constructs, or a chain that is freestanding. Thus, any overlay construct periodically should initiate a merging attempt with further overlay constructs, regardless of its size.

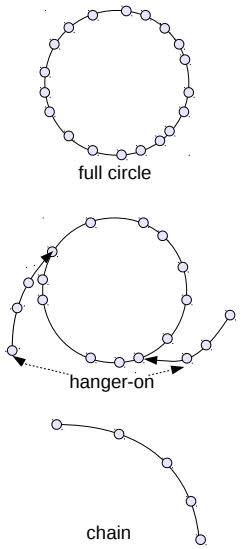


Figure 4. Possible constructs during the merging process.

For the presentation of our approach, we first describe in Section 3.1 how a single node can identify potential other nodes in other overlays. Once a potential node is identified, we discuss in Section 3.2 how the nodes in an overlay construct coordinate to limit the number of merging attempts per overlay construct. With a potential contact at hand, in Section 3.3, we present our merging protocol and point out in Section 3.4 how the merging process terminates. In Section 3.5, we extend our Ring Reunion Algorithm to initiate coordinated merge processes in parallel. Our approach, termed in the evaluation Reunion2, Reunion3, and Reunion4 using 2^2, 2^3, and 2^4 parallel instances, improves the merging performance and robustness. Finally, in Section 3.6 we discuss how our Ring Reunion Algorithm handles the parallel merging of multiple overlay rings.

3.1. Identification of New Overlays

For the merging of several unconnected overlays, at first it is necessary that at least one node in one overlay (the initiator) makes contact with a node from an other overlay (contact node) to start a merging algorithm. Finding a first single node in an existing overlay is the classical bootstrap problem in peer-to-peer overlay network. Common solutions are to use either (a) to keep track of previously know hosts (passive list), (b) to use an external/public list (active list) provided out of band, or to (c) actively search for network members.

A passive list, as proposed in the Ring-Unification Algorithm [7,32], is a simple list of known contacts that is maintained while nodes are part of an overlay network. Each node keeps a list of node contacts (IP address, port, node ID) associated with nodes that were encountered. Nodes which are detected to be offline are moved from the routing table into the passive list, where they are periodically tested for reachability. Once nodes in the passive list are detected to be reachable again are removed from the list and are considered as possible contact nodes from a separated overlay that is to be merged. The disadvantage of using the passive list is that only nodes from the original routing table are considered to be mergeable nodes. In case of network partitioning events only those nodes can be found, which have been known previously.

In order to avoid this restriction, we introduce an alternative procedure to find new or separated overlays: a public list of randomly chosen nodes is maintained individually at each participating node. The public contact list itself is obtained by every node from the bootstrap node during the join process.

We decided to limit this public contact list to a maximum of 160 nodes which is equal to the number of finger entries in our Chord simulations. Additionally, the public list of a

specific node n could be updated with nodes which want to join the overlay and therefore visit node n as the first-contact node. IP-scans could be used to search actively for further separated overlays. Similar to the passive list, nodes on the public list are periodically probed to be reachable and considered as mergeable contacts. In contrast to the passive list, nodes are not removed from the public list if they are found to be reachable. Only in the case of a size-limited list, nodes might be replaced by other (better suited) nodes. Within our evaluation, we compare the public list approach to the behavior of the passive list without updating the public list, since the number of participating nodes is clear during simulations.

3.2. Coordinating the Merging Attempts in a Construct Through a Merging Probability

In order to limit the number of merging processes to one per individual construct, instead of one per node, we introduce a certain probability for starting merging processes. Specifically, if a node is member of an overlay construct of size $size_r$, then each participating node's probability to initiate a merging process is $\frac{\alpha}{size_r}$, where parameter Alpha α defines the number of started merging operations. Thus, within a time frame any separated overlay aims at finding new contacts and initiating α merging processes. If, for example, the probability is given by $1/size_r$, only one node per ring starts a merger instance on average.

While α is a parameter, for which we specify good values in the evaluation, the size of the ring $size_r$ has to be calculated spontaneously. In the past, we have proposed several overlay statistic monitoring approaches such the tree-based over-overlay SkyEye [33] or Gossip-based approaches [34] that require more interaction with neighboring nodes (and might be hindered in a partitioned network). Optimistically, a good estimation for the size of the network is also sufficient. Thus, we opt for a quick, locally computable estimation on the network size by following an approach similar to that presented in [35] and estimate the size of the network by calculating the average responsibility range within the known contacts of a node. Therefore, we consider the id ranges which are managed by the predecessor, successor(s), and fingers of a node. Based on this information, we estimate the average responsibility id range, $resp_p$, of a node. Consequently, the estimated number of nodes in the overlay is given by $size_r = \frac{2^{160}}{resp_p}$, i.e., the size of the identifier space of the network divided by the estimated average responsibility range of a single node.

Thus, any node initiates a merging process with probability $\frac{\alpha}{size_r}$ within a given time interval. As a result, each overlay construct initiates approximately α merging processes within this time interval. As shown in the evaluation, the Ring Reunion Algorithm performs well with this concept.

3.3. Merging Algorithm

If a contact node is found and the probability condition is fulfilled, then the merging protocol, as depicted in Figure 5, is started. The initiator starts the merging algorithm by sending the contact details of its successor to its alternative successor (if $altSucc \in (n, succ)$, line 10, Figure 5), or by sending the contact details of the alternative successor to its successor (if $altSucc \notin (n, succ)$, line 13, Figure 5).

The receiver of this merge message considers the contact information, which have been sent with, as its own alternative successor. The sender is seen as possible predecessor of the receiver and is used as new predecessor, if it fits better than the current predecessor. By this, the active node (receiver) is able to combine the two overlay constructs locally by deciding which contacts are best suited to be kept in its routing table. This active role of combining two rings in this specific point of the id space is termed as having the merger token. Next, the successor with the next closest ID (to the active node) is informed about potential new candidate nodes and, thus, the merger token is passed on to it, so that it can

update its successor and predecessor. By this process, a node for a node and a successor for a successor, the two overlay constructs are merged.

```
1  receipt of STARTMERGER(contact) from m at n
2     sendto contact : LOOKUP (n.id)
3     if altSucc is received from contact within time γ then
4        MERGE (altSucc)
5     end if
6  end event

7  receipt of MERGE (altSucc) from m at n
8     pred := m
9     if n ≠ altSucc then
10       if altSucc ∈ (n, succ) then
11          sendto altSucc : MERGE (succ)
12          succ := altSucc
13       else
14          sendto succ : MERGE (altSucc)
15       end if
16    end if
17 end event
```

Figure 5. Ring Reunion Algorithm.

In contrast to Chord–Zip, the Ring Reunion Algorithm does not combine finger entries to prevent current routing behavior. For the Ring Reunion Algorithm, an initial lookup is needed to start the merger. After that, the merger token is passed through the ring, once for each node, until the algorithm terminates. Consequently, the time complexity for merging is given by $O(N + log\ N)$, where N denotes the number of nodes participating in the overlay.

3.4. Termination

The Ring Reunion Algorithm terminates if the received alternative successor is equal to the node which holds the merger token (line 9, Figure 5). That is, the node has been commanded to merge itself. If, then, one node obtains a contact node from the own ring, a lookup is started in this ring to find the successor node of the node's id. Consequently, the node receives itself as the successor node for its id, whereupon the merger instance is terminated. Through this behavior, the Ring Reunion Algorithm recognizes if a given contact node is already merged and is able to stop immediately. In Figure 6, we give an example: node n came in contact with a node from its own ring and asks it for its alternative successor (1). The contact node starts a lookup to find n's alternative successor (2–3). The alternative successor is then given to n (4). Node n tries to merge the given node and stops immediately, as it is equal to the given alternative successor (5–6). If one message is received by a node for the second time, e.g., due to timeout and retransmission, the second merger instance terminates quickly, as the first merger instance (received via first message) has already merged the ring locally.

3.5. Parallelization

Inasmuch as we focus on merging algorithms which act locally in a ring without requiring knowledge about the global ring, these mergers can be improved by starting multiple merger instances simultaneously and in parallel. While this drastically improves the merging speed, it is very important that all merger instances terminate eventually and do not hinder each other in their performance.

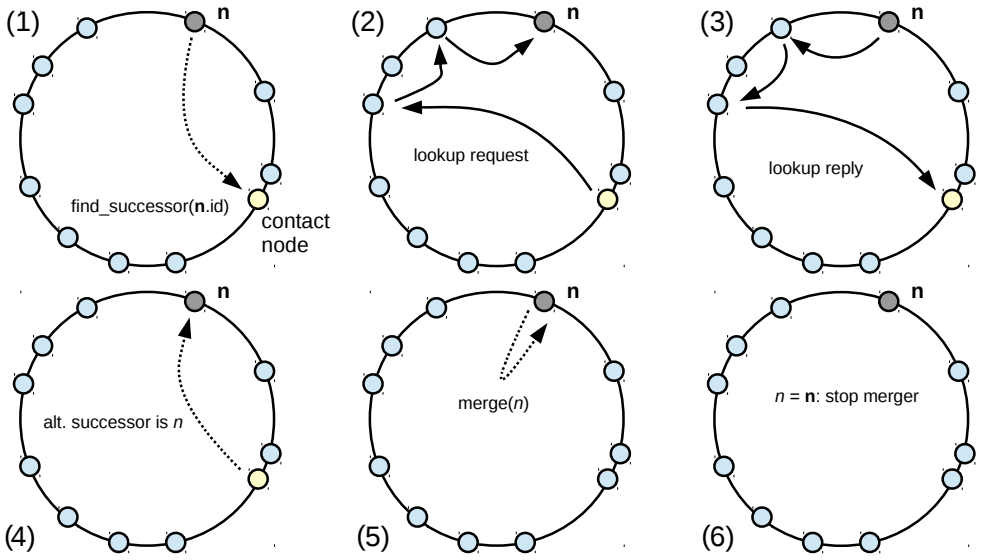


Figure 6. Node tries to merge own ring. Merger stops immediately when it receives itself as contact.

To improve the Ring Reunion Algorithm, we present an extension of the algorithm to start multiple merger instances simultaneously and independently (see for the extended Algorithm the Pseudo Code in Figure 7). The method DISTRIBUTE can be started every time a merger instance is started (line 2). As a result, the initiator node informs its furthermost finger contacts to initiate a new merger instance, see also Figure 8. Then, those finger contacts inform their second furthermost finger contacts, respectively; this continues, and thereby, the information to start new instances is equally distributed among all nodes in the initiator node's ring (lines 7–10). These nodes are informed about the presence of a second overlay construct and start a merging attempt in this second overlay upon receiving related contact information. Concurrently, a distributed counter is decreased with each message, so that exactly $2^{\lambda} - 1$ additional merger instances are initialized, where λ is a fixed parameter (line 7).

Hereafter, all nodes that have received a DISTRIBUTE message start the MERGE method (line 11) and begin to merge a given ring (lines 14–24). Figure 8 demonstrates that, with each step, i, 2^i nodes are asked to start a merger instance, so that after 3 steps, $2^3 = 8$ instances are started. The terms *Reunion(i)*, e.g., Reunion2, Reunion3, etc., in our evaluation correlate to the exponent i.

3.6. Example of Merging of Several Rings

As the example in Figure 9 highlights, the Ring Reunion Algorithm is capable of merging multiple rings simultaneously. It terminates properly and creates one single ring. Figure 9 shows a scenario in which two nodes try to merge one node (27) at the same time. Since one MERGE message will always be received first, the second incoming Ring Reunion instance will always merge a ring that has been merged before. In this example, node 11's MERGE message is received by node 27 shortly after node 18's MERGE message has arrived at node 27. Arrows describe forwarded MERGE messages, and solid lines denote the successor pointers from each node to its successor.

```
1  receipt of STARTMERGER(contact) from m at n
2     DISTRIBUTE (0, contact)
3  end event

4  receipt of DISTRIBUTE (cnt, contact) from m at n
5     sendto contact : LOOKUP (n.id)
6     if altSucc is received from contact within time interval γ then
7        while (cnt < λ) do
8           sendto fingerEntry(lastEntryPos − cnt) : DISTRIBUTE (cnt + 1, contact)
9           cnt := cnt + 1
10       end while
11       MERGE (altSucc)
12    end if
13 end event

14 receipt of MERGE (altSucc) from m at n
15    pred := m
16    if n ≠ altSucc then
17       if altSucc ∈ (n, succ) then
18          sendto altSucc : MERGE (succ)
19          succ := altSucc
20       else
21          sendto succ : MERGE (altSucc)
22       end if
23    end if
24 end event
```

Figure 7. Ring Reunion Algorithm with parallelization.

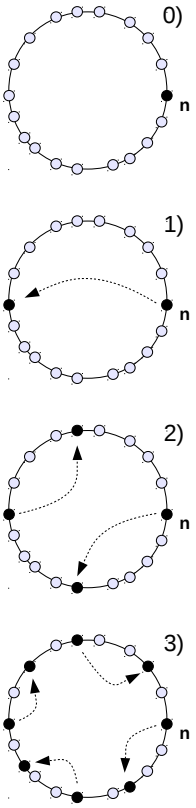


Figure 8. Distribution algorithm: furthermost finger contacts are invited to start instances of the merging algorithm as well.

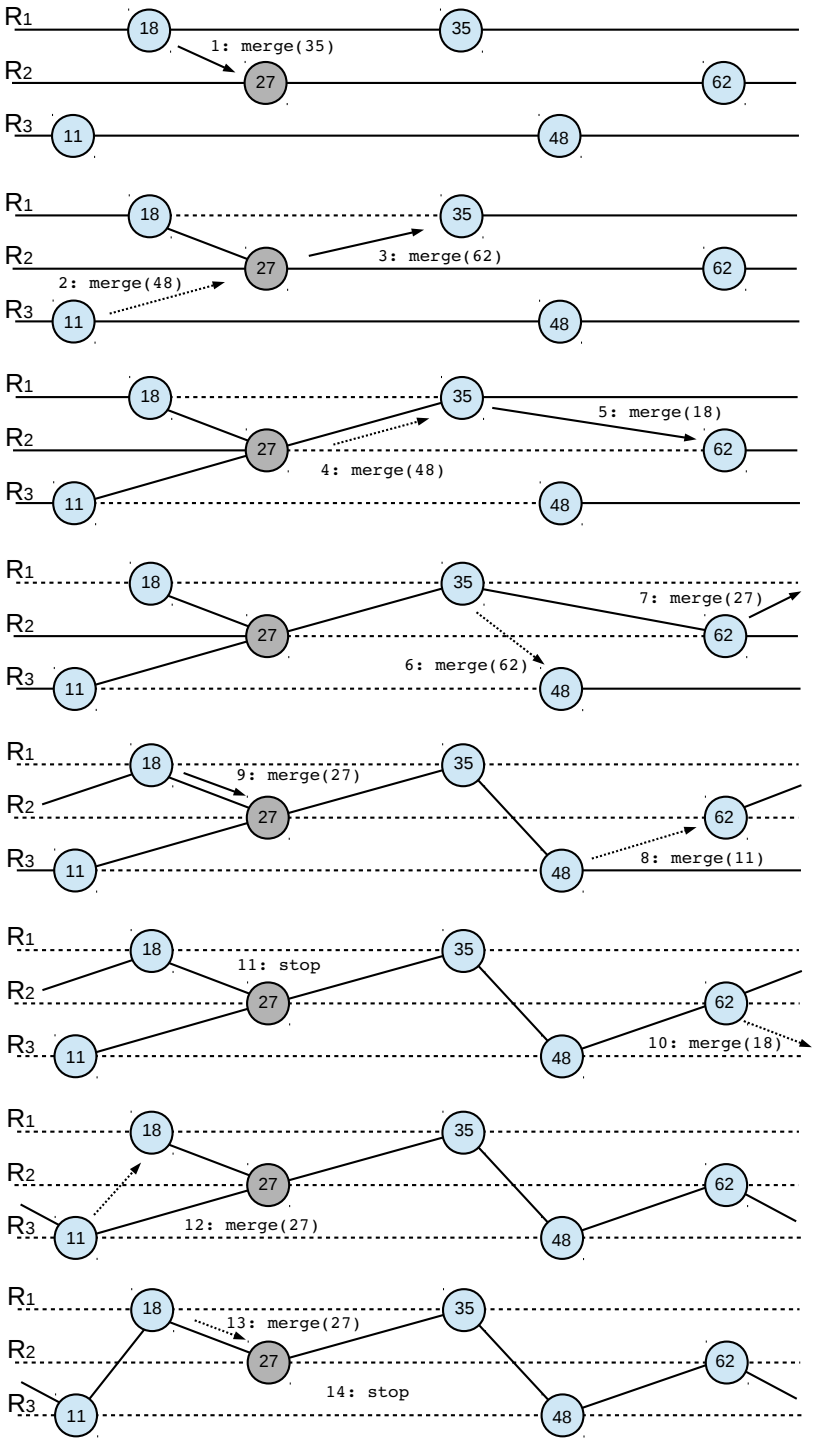


Figure 9. Example of two nodes merging the same ring, using the Ring Reunion Algorithm.

4. Evaluation

In this section, we evaluate the behavior of our proposed Ring Reunion Algorithm, both in the unparallelized (Reunion) and parallelized version (Reunion2, Reunion3, and Reunion4) in comparison to Chord–Zip [6] (Chord–Zip) and Shafaat et al.'s [7] simple Ring Unification (Simple) and Gossip-based Ring Unification (Gossip).

4.1. Evaluation Goals—Metrics

We specify criteria which explain the performance and correctness of a single merger algorithm. First of all, a merger should be able to merge multiple overlay constructs in

a certain time, meaning that multiple constructs should be combined in such a way that one global ring is created as a result. Second, a merger should be capable of arranging all successor pointers in a proper way, since this is the most important criterion that routing within a ring is possible. Finally, the costs in terms of traffic are to be evaluated to decide whether the overhead of the approach is acceptable. Next, we discuss the metrics in detail.

Number of Constructs:

With this metric, the current number of constructs is observed in order to determine the number of isolated communication islands in the overlay. To ascertain the number and types of constructs, we maintain in our evaluation environment information about peers and their successor pointers. Each connection between peers is then traversed in order to detect present constructs.

Correct Pointers:

For better evaluation of any merger algorithm, we determine the number of correct successor pointers at a specific time. Therefore, we extended the analyzer in our evaluation environment in the way that each successor pointer is periodically compared with the value it should contain in a single global ring. Consequently, this metric describes perfectly the fraction of current correct pointers in comparison to overall correct pointers.

Traffic Overhead:

In order to judge the quality of the presented algorithms, it is also necessary to quantify the overhead in terms of message and bandwidth consumption. Ideally, the traffic overhead of an merging approach should be in relation to the number of constructs in the underlying network. With one large ring, the overhead should be minimal; meanwhile, with more rings and constructs, the contact searching and merging overheads are acceptable, and can rise.

4.2. Simulation Setup

For our simulations, we use PeerfactSim.KOM [36–38], an event-based simulator for peer-to-peer protocols, in order to obtain realistic results and insights on our research. Each simulation uses Chord and has been run with 10 different random seeds, so that all values in the graphs represent the average over 10 runs. Each algorithm has been tested separately and independently from other merging algorithms. In addition, each simulation uses GNP coordinates [39] to estimate delays in the fundamental network realistically. The network layer was extended to strictly isolated dedicated regions, any connection in or out of selected regions, which are dropped in selected times. Reasonable approximations for jitter and message delay are integrated into the simulator by using measurements from the PingEr project [40]. Except in Scenario E, we do not consider churn and packet loss, as both attributes might obscure the characteristic behavior of a specific merging algorithm. We present our simulation setup in Table 1.

In Setup A.1, we investigate the performances of various merging algorithms with 1024 nodes in total, while merging three overlay networks without any network partitioning events. This scenario should reveal whether each algorithm is able to unify multiple separated Chord rings. We want to filter bogus mechanisms out and obtain a first feeling of how each algorithm performs under simple conditions. Thus, at the beginning of this scenario, three different Chord rings are created with 341, 341, and 342 nodes, respectively. Within the 10th minute, two nodes, each selected from one of the two rings with 341 nodes, start to merge one and the same contact node from the 342-node ring. After 180 min, the simulation is finished.

In Setup B.1, we compare the performance of the most promising approaches from A.1 in a large-scale network with 10,242 nodes. Two Chord rings are formed at the beginning of

the scenario. In the 10th minute, a contact node from one ring is given to the initiator node in the other ring, so that a merging procedure is started. The simulation is finished after 180 min. The Gossip-based Ring-Unification Algorithm is tested with a fanout parameter of 4, since this value is suggested by the algorithms authors in [7] This setup finally shows that our Ring Reunion Algorithm and its parallelized versions Reunion2, Reunion3, and Reunion4 (4, 8, and 16 instances) are superior in terms of merging reliability and merging speed. Within this setup, we also observe the effect of different fanout settings for the Ring Reunion Algorithm.

In Setups C.1–5, we evaluate approaches for identifying new nodes for merging. The simulations of Setups C.1-2 begin with a join phase. After 150 min 1024 nodes have built a global Chord ring. In the 180th minute, a group of 310 nodes is, due to an isolation event, separated from the other nodes. During the isolation, nodes in both separated regions search for other reachable nodes in order to start the merger algorithm. In minute 240, the isolation is canceled so that all nodes are reachable again. After 360 min, the simulation is stopped. Besides the passive list in Setup C.1, we consider a public list of potential contacts in Setup C.2 that is frequently updated. To reduce the quantity of messages sent by the merging algorithms each node only starts a merger if it picks a random value less or equals $\alpha / size_c$, where $size_c$ is the estimated number of nodes in the current construct.

In Setups C.3–5, we choose values of 1, 5, 10, and 100 for α, i.e., the number of aimed ongoing merge activities in a network. Similar to the previous scenario a Chord ring with 1024 participants is formed from minute 0 to minute 150. After 180 min a group of 400 nodes is isolated, after 240 min, it is connected again to the remaining groups, and after 360 min, the simulation is finished. Setups C.3–5 help in finding suitable values for the parameter α, which relates to the probability with which a node initiates a merging approach.

In Setup D.1, we evaluate the Ring Reunion Algorithm with four parallel instances and ideal $\alpha = 10$ (estimated in Setups C.3–5) against the simple and Gossip-based Ring-Unification Algorithm in a complex and more realistic scenario in which multiple regions are isolated. We examine this scenario with $\alpha = 1$ and $\alpha = 10$ to compare a poor value for α (1) to a fair one (10). As described in the previous scenario setup, 1024 nodes join a common Chord ring during the first 150 min. A group of 400 nodes is then isolated from the 180th minute to the 240th minute. In addition, another group of 50 nodes is isolated from the 200th minute to the 240th minute. Finally, from minute 240 to 300, a third group of 100 nodes is isolated from the other regions.

Finally, in Setups E.1-2, we consider churn; a group of 400 nodes is separated from the rest of the Chord ring in minute 180, when all nodes have joined the network successfully. The duration of the isolation lasts 60 min. From minute 240 to 360, when the simulation finishes, the nodes unify the partitioned network again. In the previous scenarios, we did not consider churn in order to focus on the fundamental behavior of each tested algorithm. This time, churn is enabled throughout the whole simulation to show that our Ring Reunion Algorithm is able to handle it without loss of performance. In addition, with this last simulation setup, we examine the influence of different parameter settings on the Ring Reunion Algorithm in the presence of churn. The supplementary parameters we examine in this scenario are the parameter that controls the number of parallelized merger instances and the interval within which the public list is iterated.

Table 1. Simulator setup and scenario setups.

General Settings	
Simulator details	PeerfactSim.KOM [36–38], 10 seeds per setup
Network model	GNP [39], jitter based on [40], no packet loss
(A) Simultaneous Merging of Three Networks	
Algorithms	Chord–Zip [6], Simple Ring Unification [7], Gossip-based Ring Unification [7], Ring Reunion (1 and 4 instances), no churn
A.1 Three rings	0 min: setup of 1024 nodes in three Chord rings (A: 341, B: 341, and C: 342 nodes). From 10 min: merge algorithm started by passing random nodes from A and B to one node in C. 180 min: simulation end.
(B) Direct Comparison of Gossip-based Ring Unification [7] and Ring Reunion with Various Instances	
Algorithms	Gossip-based Ring Unification (4 + 4 instances), Ring Reunion (4, 8 and 16 instances), no churn
B.1 Two large rings	0–150 min: setup of 10,242 nodes in two Chord rings (each with 5121 nodes). From 10 min: merge algorithm started by adding one connecting link. 180 min: simulation end.
(C) Testing of Various Approaches to Identify Contacts in Other Overlays and to Start the Merging Process	
Algorithms	Chord–Zip [6], Simple Ring Unification [7], Gossip-based Ring Unification [7], Ring Reunion (1 and 4 instances), no churn
Scenario	0–150 min: 1024 nodes join two different Chord rings (each with 512 nodes). From 180 min: 310 nodes (in one country) are isolated. 240 min: isolation is canceled, all nodes are reachable. 360 min: simulation end.
C.1 Passive list	Maintaining a list of previously online nodes, which is checked every 3 min. If active node is found, start merging process.
C.2 Public list	Publicly known 160 contacts are tested every 4 min in iteration. Once an active contact is found, the merging process is started with probability 1.
C.3–5 Public list, dynamic probability: Gossip, Reunion, Reunion2	Nodes maintain a list of 160 nodes which is obtained from bootstrap node during join process. Checking online presence of a random node in this list every 4 min. Start merging process with probability of $\alpha / size_r$, where $size_r$ is the estimated number of nodes in the current construct.
(D) Complex Scenario Using the Public List Approach	
Algorithms	Gossip-based Ring-Unification Algorithm [7] and the Ring Reunion Algorithm (1 and 4 instances), public list with $\alpha = 10$, no churn
D.1 Complex rings	0–150 min: setup of 1024 nodes in one Chord ring. Group of 400 nodes is isolated in 180–240 min. Group of 50 nodes is isolated in 200–240 min. Group of 100 nodes is isolated in 240–300 min. 360 min: simulation end.
(E) Parameter Studies and Churn	
Algorithms	Ring Reunion (8, 16, 32, 64 instances), public list, churn enabled
Scenario	0–150 min: 1024 nodes join Chord ring. From 180 min: 400 nodes become isolated. 240 min: isolation is stopped, all nodes are reachable. 360 min: simulation end.
E.1 Changing α and Instances	Algorithm is combined with $\alpha \in \{10, 100\}$. The interval with which the *public list* selects a merging candidate is set to 5 min.
E.2 Changing frequency of merge attempts	Algorithm combined with intervals of 5, 10, 15, and 20 min in which merger instances are started, to find compromise between operation time and bandwidth consumption.

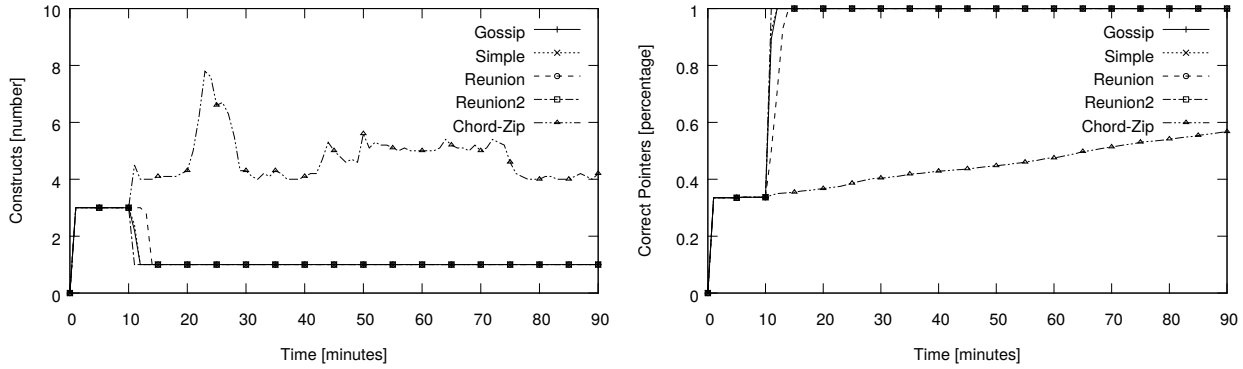
5. Evaluation Results

In this section, we present the simulation results obtained from simulating the scenarios described earlier. First, in Section 5.1, we test the basic functionality of Chord–Zip, the basic and the Gossip-based Ring-Unification Algorithm and our Ring Reunion Algorithm by merging three Chord rings simultaneously. In Section 5.2, we directly compare the performance of our Ring Reunion Algorithm with the Gossip-based Ring-Unification Algorithm. Section 5.3 presents the results for Setups C.1–5 in which we test different approaches to identify contact nodes from other constructs. We test the behavior of our Ring Reunion Algorithm in comparison to the simple and Gossip-based Ring-Unification Algorithm during the isolation of multiple regions in Section 5.4. To conclude our studies, we investigate our Ring Reunion Algorithm in the presence of churn in Section 5.5.

5.1. Evaluation Results for Setup A.1: Simultaneous Merging of Three Networks

In Setup A.1, three different Chord rings are created, with 341, 341, and 342 nodes per ring, respectively. In two of the three rings, one node is selected to start the merging procedure. Thus, two nodes start to merge the third ring simultaneously. Our first goal is to determine which of the presented algorithms can merge two and more rings simultaneously without the presence of churn or network partitioning events. As we see later, this ability turns out to be important if multiple instances are started automatically whenever a present contact node is detected.

In more complex scenarios, it happens that different rings try to merge another ring at the same time—for example, if two regions become connected again after a network partitioning event. Figure 10 shows the simulation result of three Chord rings which have been merged within Setup A.1. It can be seen that both the Ring-Unification Algorithm and our Ring Reunion Algorithm are capable of merging multiple rings without major effort and in similar time. In Figure 10b, one can see that the Chord–Zip algorithm needs more time to adjust the successor pointers than the other merging algorithms. Figure 10b also reveals that in more complex scenarios the Chord–Zip algorithm rearranges the successor pointers in a very slow tempo that would exceed the simulation time. Chord–Zip is much slower than the other solutions due to how it chooses its alternative successor pointers. While the Ring-Unification Algorithm and the Ring Reunion Algorithm preserve the correct order of nodes they merge, alternative successor pointers in Chord–Zip are always taken from the other ring and are therefore often not the best choice. Figure 10b reveals that the Ring-Unification Algorithm and the Ring Reunion Algorithm are capable of adjusting all successor pointers in complex scenarios within a short period of time.



(**a**) A.1 three networks, constructs. (**b**) A.1 three networks, correct pointers.

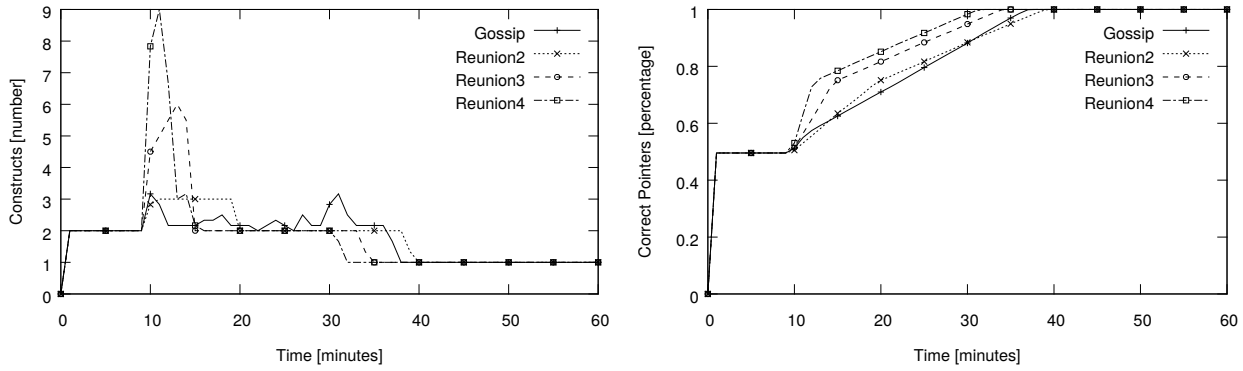
Figure 10. A.1 Merging of three networks.

5.2. Evaluation Results for Setup B.1: Comparison of Gossip-Based Ring-Unification Algorithm and Ring Reunion Algorithm

Setup B.1 is similar to Setup A.1 in the sense that two Chord rings, each with 5121 nodes per network (5120 nodes + 1 initiator), are formed at the start of the simulation. In minute 10, one node starts to merge with a contact node inside the opposite ring. As Chord–Zip is too slow to merge multiple Chord rings in an acceptable interval of time, we directly compare the Gossip-based Ring-Unification Algorithm with a fanout parameter of 4, i.e., the number of initiated mergers in each direction, to the Ring Reunion Algorithm with 4, 8, and 16 parallel instances, since those algorithms have turned out to be the fastest.

The results of Setup B.1 are shown in Figure 11. The Ring-Unification Algorithm's advantage is that it merges a ring in two directions, clockwise and anti-clockwise. Therefore, the Gossip-based Ring-Unification Algorithm with a fanout parameter of 4 (4 instances clockwise + 4 instances anti-clockwise) can be compared best to the Ring Reunion Algorithm

with 8 parallel instances (Reunion3), which outperforms the Gossip-based Algorithm. Nevertheless, the number of messages which are sent with a high number of merger instances is almost the same as the number of messages with a small number of instances.



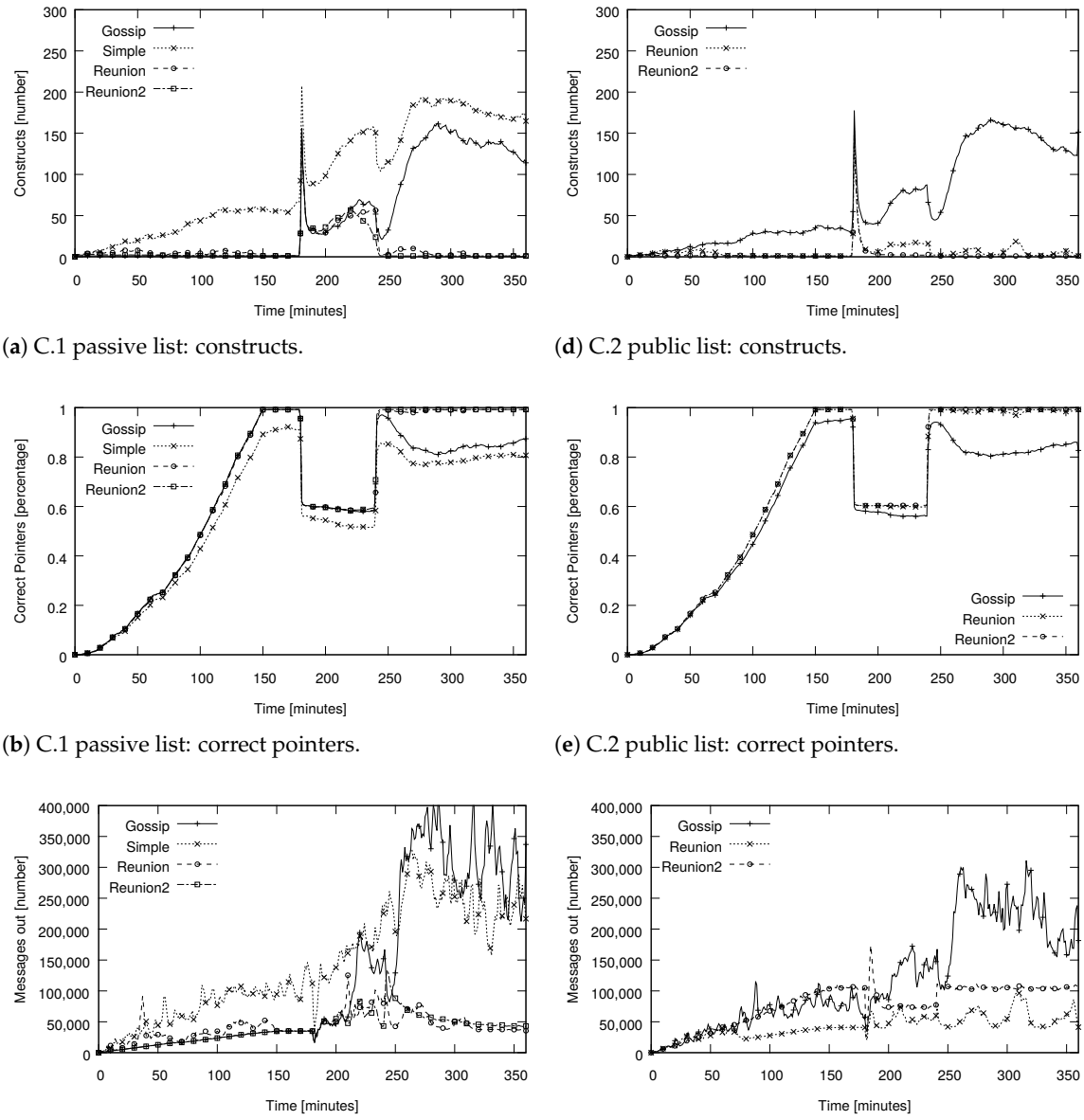
(**a**) B.1 constructs. (**b**) B.1 correct pointers.

Figure 11. B.1 Direct comparison of Gossip-based Ring-Unification Algorithm and Ring Reunion Algorithm applied in two large networks. Merging algorithm is manually started at only one node.

5.3. Evaluation Results for Setups C.1–5: Testing Various Approaches to Start Merging Instances

The scenario of Setups C.1–5 is as follows: 1024 nodes join one Chord ring. After the join phase and some additional time in which the Chord ring should have stabilized, a group of 310 nodes (C.1–2) or 400 nodes (C.3–5) is isolated from the network and therefore separated from the global Chord ring. With respect to Setup C.1 and C.2, we investigate the effect of different approaches, namely the passive and the public list, to identify the separated nodes again. In Setups C.3–5, we reduced overheads in terms of message complexity by introducing a probabilistic model to start merging procedures.

In Figure 12a–c, the usage of a passive list is shown. Nodes which are detected by a node to be suddenly unavailable are added to its passive list, which is iterated periodically to find possible contact nodes to merge. The large number of constructs observed for Chord–Zip indicated that the Chord–Zip algorithm is not suitable for being combined with the passive list, as it is too slow to react on multiple, simultaneous opportunities to start different merger instances. Surprisingly, both Ring-Unification Algorithms have not been able to handle the network isolation event within the simulation time as can be seen in Figure 12a. Only the Ring Reunion Algorithm has been able to adjust all successor pointers directly after the partitioning event finished at minute 240 (Figure 12b). Figure 12c shows that the quantity of message consumption of the Ring Reunion Algorithm does not exceed the amount of messages which are sent by Chord itself. On the contrary, the other merging algorithms produce high numbers of messages, since multiple merger instances are started to merge the high number of constructs which are formed after the network partitioning. As an alternative to the passive list, we tested a list of 160 randomly chosen and publicly known nodes which are obtained by each node from a bootstrap server during the join phase. This list is iterated periodically in Setup C.2, in order to obtain possible contact nodes.



(**a**) C.1 passive list: constructs.

(**d**) C.2 public list: constructs.

(**b**) C.1 passive list: correct pointers.

(**e**) C.2 public list: correct pointers.

(**c**) C.1 passive list: message overhead.

(**f**) C.2 public list: message overhead.

Figure 12. Comparison of passive list (C.1, left) and public list (C.2, right) for discovering overlays during partitioning events. Nodes initiate a merge when a contact node is reachable.

Looking at the usage of the public list, as Figure 12e indicates, only the Ring Reunion Algorithm manages to adjust all successor pointers within the simulated time. Again, the Gossip-based Ring-Unification Algorithm forms a large number of constructs after all network partitions have become reachable again—see Figure 12d. It might be possible that the Gossip-based Ring-Unification Algorithm is capable of merging the separated overlays again, but that would take a long time. Figure 12d,f show that the message consumption of both algorithms is highly related to the number of constructs in the underlying network.

At first glance, it might seem questionable why the public list, which actively repeats to contact the same overlay nodes again and again, should be considered as alternative to the passive list (reactive approach). The answer is very simple: the problem with the passive list is that only nodes which have been known before a network failure occurs can be considered as target nodes to merge. The passive list is limited by each node's routing table, i.e., the finger table and successor list. The public list on the opposite contains randomly selected nodes which are not dependent on the routing table. This principle

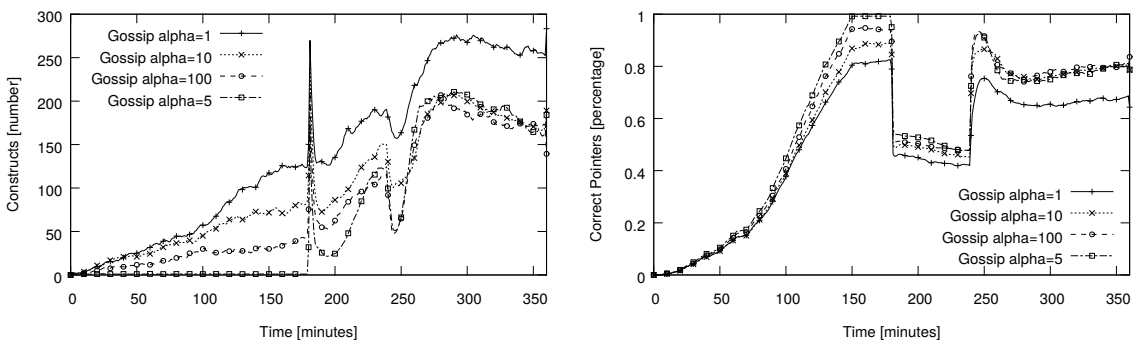
counteracts the effect of unintended group forming, stated in Section 1.1, by enriching the routing table of each node with a pool of further, well distributed contacts.

A combination of both approaches is possible but not considered in our evaluation since we aim at a direct comparison of both approaches. If long-term network partitions are considered, the behavior of the passive list is comparable to those of the public list: nodes in the routing table which are detected to be unreachable are held in the passive list, unchanged for a long time, but without the diversity of the public list. We limited the list of public nodes to 160 contacts since Chord's routing table has the same size in our simulation. In principle, this list could be periodically obtained from a bootstrap server.

We decided to test the merger algorithms with a static list which does not change during the simulation, and is not updated and fresh, in order to simulate a worst-case scenario in which no bootstrap server is available once a node joins the network. A comparison of Figure 12c,f shows that the message overhead of the public list (active approach) is comparable to the overhead produced by the passive list (reactive approach).

Next, in Setup C.3-5, we reduce the amount of messages by reducing the number of instances that are started by a specific merging algorithm. Therefore, we extended the algorithm with the ability to estimate the size of the current construct a node is in. Now, each node picks a random number out of $[0,1]$ and starts a merger instance only if the chosen random number is less than $\alpha/size_r$, where α constitutes the number of started mergers per overlay construct and $size_r$ is the estimated number of nodes in the current construct.

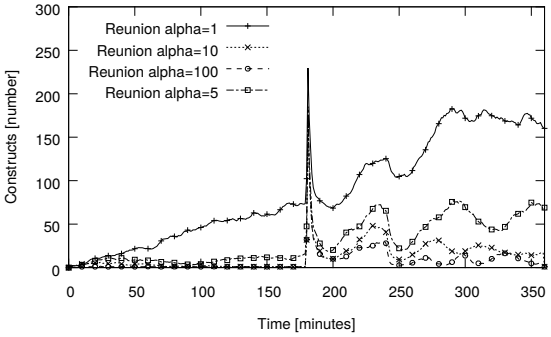
These setups and corresponding results in Figure 13 allow us to determine a good value for α, i.e., the number of merger instances per construct. As can be seen in Figure 13a, the Ring-Unification Algorithm performs poorer when fewer merger instances are started. With $\alpha = 1$ up to 250 constructs are formed, i.e., disconnected network parts; meanwhile, higher α values quickly reduce the number of constructs. Correspondingly, the ratio of correct pointers also improves, as seen in Figure 13b. The results indicate no distinct difference between $\alpha = 5$ and $\alpha = 100$. In general, the Gossip-based Ring-Unification Algorithm is not very convincing, as a complete merge is not achieved. Considering Figure 14b, we see that our Ring Reunion Algorithm performs well if approximately 10 instances per construct are started; in contrast, $\alpha = 5$ only leads to 82% correct pointers and several constructs remain (as seen in Figure 14a), and with $\alpha = 10$, only a few constructs remain. When we enable $2^2 = 4$ initiated mergers in parallel in Reunion2, the merging performs flawlessly with $\alpha = 5$ and higher α values, as the number of constructs reaches 1 in Figure 15a and we attain the ratio of 100% correct pointers that is depicted in Figure 15b. Furthermore, we learn from this evaluation result that it is necessary to react quickly on partitioning events to perform well and to reduce costs.



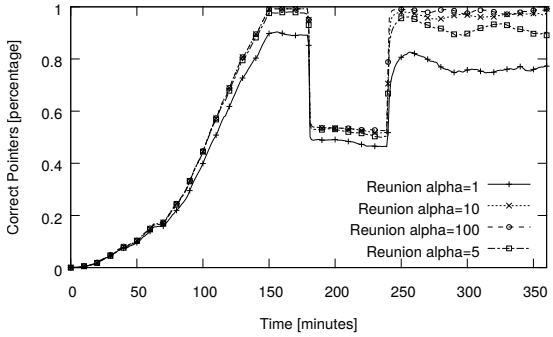
(**a**) C.3 public list, Gossip: constructs.

(**b**) C.3 public list, Gossip: correct pointers.

Figure 13. C.3 parameter study: Gossip-based Ring-Unification Algorithm Reunion3 ($2^3 = 8$ instances). Each overlay construct initiates α mergers within a time period of 4 min.

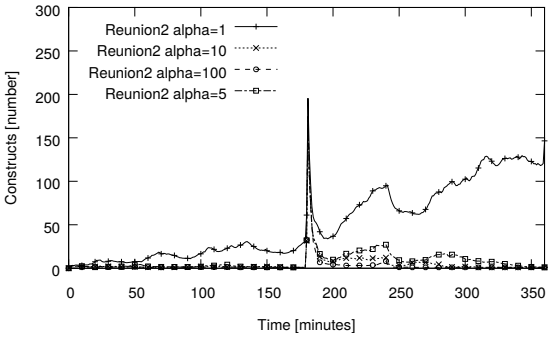


(**a**) C.4 public list, Reunion: constructs.

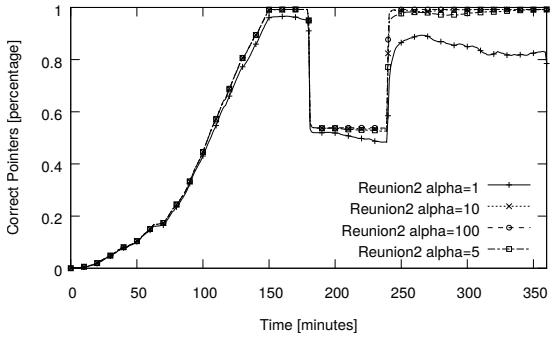


(**b**) C.4 public list, Reunion: correct pointers.

Figure 14. C.4 parameter study: Ring Reunion Algorithm. Each overlay construct initiates α mergers within a time period of 4 min.



(**a**) C.5 public list, Reunion2: constructs.



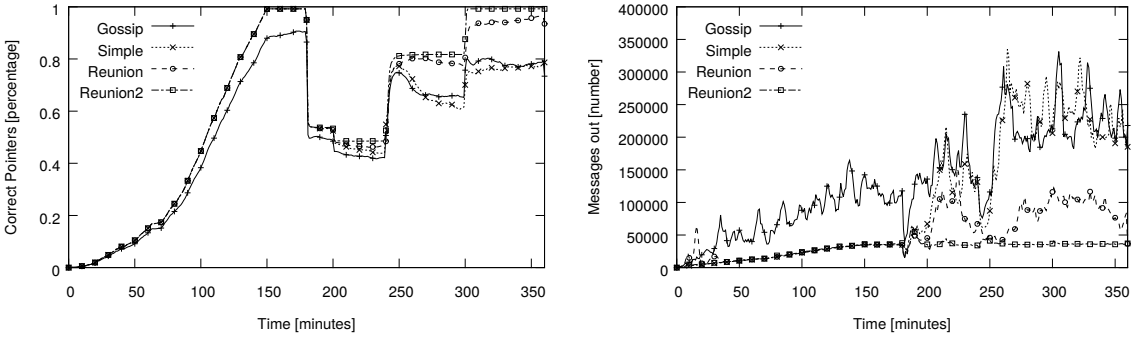
(**b**) C.5 public list, Reunion2: correct pointers.

Figure 15. C.5 parameter study: parallelized Ring Reunion Algorithm (Reunion2 with $2^2 = 4$ instances). Each overlay construct initiates α mergers within a time period of 4 min.

5.4. Evaluation Results for Setup D.1: A Complex Scenario

The next setup, Setup D.1, considers a complex scenario in which multiple regions become separated due to network isolation and churn existence during the merging period. After 1024 nodes have joined a common Chord ring during the first 150 min, a group of 400 nodes is isolated from minute 180 to minute 240. Meanwhile, another group of 50 nodes is isolated from minute 200 to minute 240. A third group of 100 nodes is isolated from minute 240 to minute 300. With this setup, we want to find out if it is possible to merge other constructs but with full circles—for example, hangers-on and chains, as described in Figure 4. Due to multiple network partitioning events in this setup, the already merged parts of the network are torn apart again.

Figure 16a verifies our expectation: the Ring Reunion Algorithm with 4 parallel instances can merge all reachable regions fast enough to handle even multiple network failures. Figure 16a shows that our Ring Reunion Algorithm corrects faulty pointers quickly after a partitioning event at minutes 180, 200, and 240. In addition, one can obtain from Figure 16b that the Ring Reunion Algorithm, if configured properly, does not produce much more messages during network partitioning events than usual. In conclusion, our evaluation shows that the Ring Reunion Algorithm is fast enough to handle even complex use cases with minimal additional message complexity.



(**a**) D.1 correct pointers. (**b**) D.1 message overhead.

Figure 16. D.1 Complex, realistic scenario in which multiple network partitioning events occur. The public node list approach is combined with $\alpha = 10$ mergers per overlay construct.

5.5. Evaluation Results for Setups E.1–2: Parameter Studies and Churn

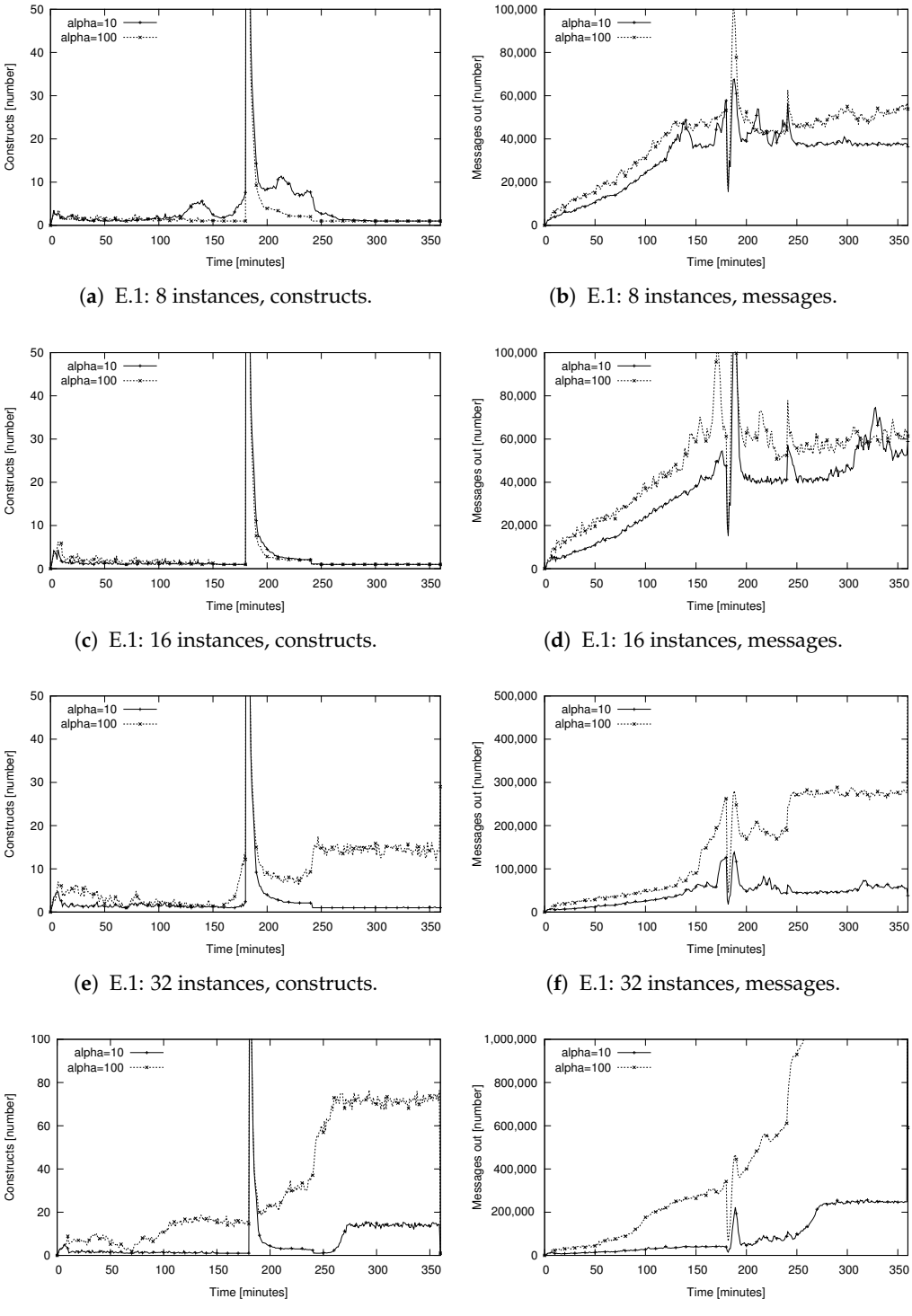
In Setups E.1–2, after all nodes have joined the network, a group of 400 nodes is isolated from the global Chord ring in minute 180. In this section, we examine the behavior of the Ring Reunion Algorithm in the context of churn and different parameter settings, in order to determine a suitable configuration of our algorithm in realistic use cases. More specifically, in Setup E.1, the interval with which the public list selects merging candidates, i.e., the interval with which merger processes are started, is set to 5 min. Value α, which regulates the number of merger attempts per construct, is set to 10 and 100. With this setting, we show that the message overhead is larger than the overall effect of the merger algorithm. In Setup E.2, we set $\alpha = 10$ and vary the interval with which merger processes are started from 5 min to 20 min in steps of 5 min. In both setups, we investigate the effect of parallel merger instances on the Ring Reunion Algorithm reliability.

Figure 17 reveals the behavior of the Ring Reunion Algorithm if multiple parallel instances are distributed by the first initiator node. In this scenario, every 5 min, the public list tests a contact node to determine how reachable it is. In Figure 17a,c, one can see that the merger algorithm is able to unify the disrupted network, no matter if 10 instances per construct are started or 100. On the contrary, if the number of parallel instances is too high, the merger algorithm operates poorly in some cases. In 50 percent of our simulations, with 64 parallel instances, the number of constructs suddenly rises after the network isolation stops, so that in the end the merger shows to be unsuccessful, as to be seen in Figure 17g. Nevertheless, Figure 17b,d,f,h prove that the message overhead produced by the Ring Reunion Algorithm depends on the number of overlay constructs and can be limited by reducing the number of merger attempts per construct by adjusting parameter α.

Figure 18 shows the results of a simulation in which multiple parallel instances have been tested in combination with different intervals for starting merger instances with $\alpha = 10$. Considering the number of constructs in Figure 18, it can be observed that high numbers of parallel instances lead to better operation, with fewer attempts being started to merge the overlay. Considering the quantity of messages, on the other hand, reveals that the number of simultaneous instances does not affect traffic overhead and the resulting bandwidth consumption.

The behavior of the Ring Reunion Algorithm can be explained as follows: if the network is not yet fully stabilized after a partitioning event, and if multiple merger instances have been started, the current overlay constructs are suddenly reordered. As a consequence the number of overlay constructs rises in this short period of time, as can be seen best in Figure 18g for all intervals greater than 5 min. Furthermore, in some cases, the determination of the additional merger instances takes more time than the actual merger process or

the interval within which new merger attempts are started. Hence, the additional started merger instances disrupt current overlay constructs again. In a few cases, this behavior leads to a dysfunction of the merger process which is caused by wrong parameter choices. Another reason for this wrong behavior is changes in the routing table, due to churn. If one node leaves the overlay suddenly, it might happen that too many instances try to unify the falsely detected overlay partition again.



(**a**) E.1: 8 instances, constructs.

(**b**) E.1: 8 instances, messages.

(**c**) E.1: 16 instances, constructs.

(**d**) E.1: 16 instances, messages.

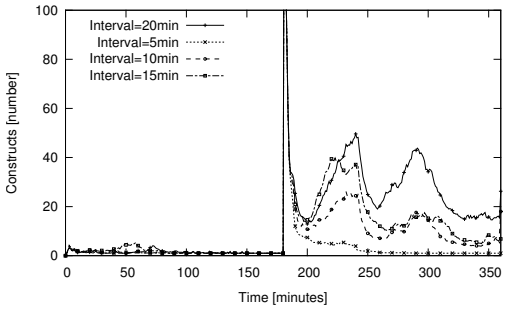
(**e**) E.1: 32 instances, constructs.

(**f**) E.1: 32 instances, messages.

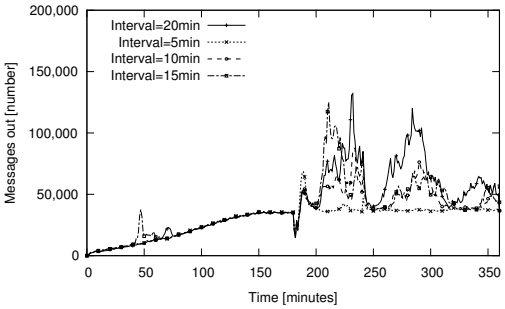
(**g**) E.1: 64 instances, constructs.

(**h**) E.1: 64 instances, messages.

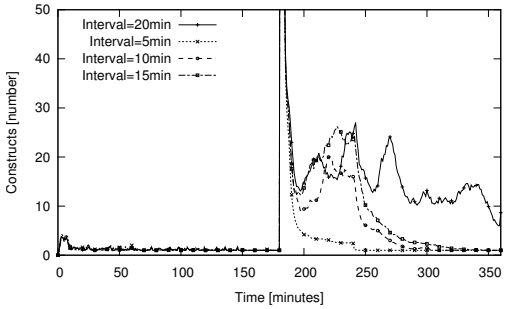
Figure 17. E.1: Ring Reunion Algorithm with parallel instances, $\alpha \in \{10, 100\}$.



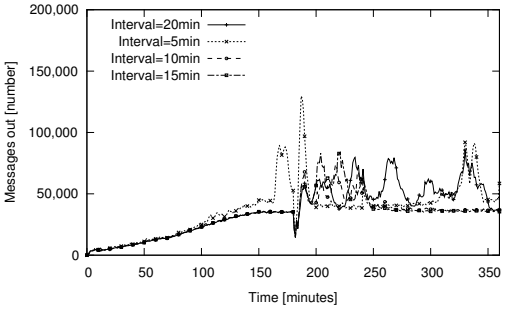
(**a**) E.2: 8 instances, constructs.



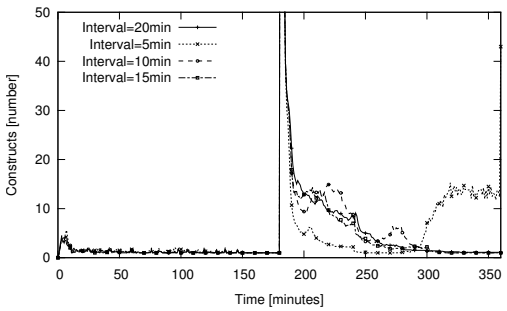
(**b**) E.2: 8 instances, messages.



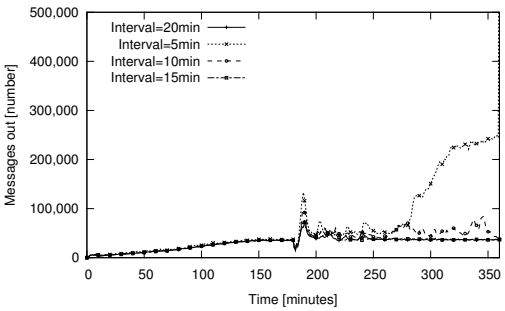
(**c**) E.2: 16 instances, constructs.



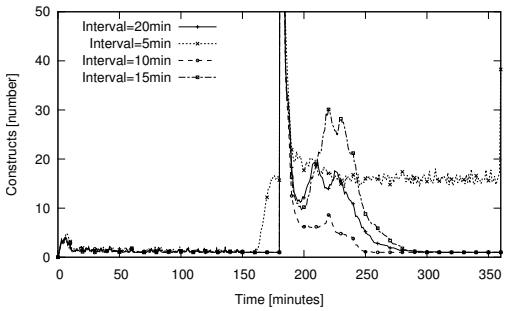
(**d**) E.2: 16 instances, messages.



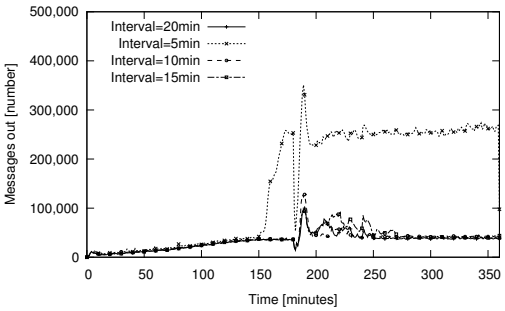
(**e**) E.2: 32 instances, constructs.



(**f**) E.2: 32 instances, messages.



(**g**) E.2: 64 instances, constructs.



(**h**) E.2: 64 instances, messages.

Figure 18. E.2: Ring Reunion Algorithm with parallel instances and various values for interval parameter, $\alpha = 10$.

To conclude our study, we suggest that a good route is to limit the number of additional merger instances so that not too many instances are created simultaneously. Although additional instances increase the speed of our Ring Reunion Algorithm, too many merging processes, started too often, can cause an opposite effect for the Ring Reunion Algorithm. Configured properly, the Ring Reunion Algorithm is able to operate fast and reliably without producing too many overlay constructs.

5.6. Discussion on Real-World Deployment

As future work, we aim to integrate an adapted version of Ring Reunion in our peer-to-peer-based online social network LibreSocial [4], which we previously tested with up to 2000 instances in [41] and that uses Pastry as overlay. Using LibreSocial as a basis will allow us to observe the actual effects of network partitioning and merging in a real-world deployment.

In our publication on Ca-Re-Chord [10], we also analyzed Re-Chord [9], and saw that, through simulation, we can identify elements of theoretical models that have been overlooked in their relevance. The same applies for real-world implementations in comparison to simulations, as we experienced when running FreePastry (in LibreSocial) compared to Pastry [3]. Real-world implementations must deal with all real networking effects being in place, while simulations and theoretical models capture only a fraction of the influencing factors.

From our work with real-world P2P applications, we know that the heterogeneity of the participating devices is a challenging issue and fault tolerance is critical. Faulty devices might fail to follow the Ring Reunion protocol, and thus the merging might stop. Typical countermeasures are to perform the process in parallel to avoid the impact of individual faulty devices or to validate the proper functionality of nodes in the overlay and mark them as faulty. A similar approach is used in Kademlia through parallel and iterative routing compared to the singular and recursive routing of early overlay networks. We investigated capacity-related roles in overlay networks in [42] and have a basis to apply this to the Ring Reunion approach in a context in which real-world investigation shows demand.

Faulty behavior is also strongly connected with malicious behavior, i.e., modified versions of P2P software that aim to disrupt the network; while at first glance the Ring-Unification approach does not seem to be able to cause harm—since, when applied in a connected ring, the ring remains as it is—a further in-depth investigation is recommended.

A further element to consider in a real-world deployment are nodes assumed to be able to connect to "random" other nodes using their active or passive list; while a P2P network is typically run by private end devices in regular households, these devices are hard to reach if they are behind a router using network address translation (NAT). In this case, it is essential that port forwarding is enabled in the router, allowing specific nodes to be reached using their IP address and port. The used port should be well known, and firewall as well as NAT settings should be configured appropriately. In [43], we already explored how an overlay can cope with a subset of nodes not being fully compatible with the protocol and only partially supporting the desired networking features; meanwhile, in "Minicamp: Middleware for Incomplete Participation in Structured Peer-to-Peer Monitoring Protocols" [43], we assume that a core structured overlay is in place and that we have an additional monitoring over-overlay that is only partially supported. In our case, we have an additional Ring Reunion approach that might be only partially supported.

Another issue in deploying Ring Reunion in a P2P network is the source of the public contact list while the network is partitioned. The core idea ultimately is to have a list of candidate node contacts to periodically scan and test whether they are available or not, regardless of whether the node information in the list is individually observed or downloaded/obtained from somewhere.

In case the Internet in a country is operational, offering and obtaining such lists, e.g., through websites, local exchange, or even scribbles on paper, is easy. In case the Internet connection between countries is shut down, and national/local Internet connectivity is given, local networks could emerge, and our Ring Reunion approach helps in supporting reconnection once the connectivity is reestablished. Contact lists could be provided on

local webservers, i.e., some local communities per country would suffice to provide these helpful lists.

Finally, in cases where all Internet connectivity is nationally stopped, only local area networks could be run, e.g., through local switch/Wi-Fi infrastructures. Transferring lists through USB sticks or addresses on paper is an option; while such local and regional networks might emerge (and later merge), opportunistic networks, which are able to cope with intermittent connectivity, are based on physical movement to bring nodes close together. They might be a better option. We explored this technology in [44,45] using the example of the Arab Spring events in Egypt in the early 2010s, when the Internet was shut down. Popular apps now include Briar (https://briarproject.org/, accessed on 26 May 2025), Reticulum (https://reticulum.network, accessed on 26 May 2025), or Guifi.net; the corresponding websites were visited in late May 2025.

In summary, the Ring Reunion Algorithm proposed in this article has been evaluated using simulations which show promising behavior in the case of simulated network partitioning. However, when adopting it in real-world applications, further factors might play a role that pose drastic challenges to address. Only through direct implementation and testing in the wild can we explore the full potential and capability of the approach.

6. Conclusions

In this paper, we address the issue of peer-to-peer overlays in the presence of dynamic Internet partitioning on national scale. We show that Chord [2], as well as previously proposed merging algorithms for ring-based overlays such as Chord–Zip [6] and the Ring-Unification Algorithm [7], are incapable of reliably merging several Chord overlays under the considered scenarios. While Chord–Zip already fails to merge more than two overlays in parallel and in reasonable time, the Ring-Unification Algorithm has its problems in merging overlays after heavy network partitioning events.

We present in this paper a novel merging algorithm for ring-based peer-to-peer overlays, named the Ring Reunion Algorithm. The Ring Reunion Algorithm has been evaluated using either a public list of 160 randomly selected online contacts, which is used to select potential nodes for a merging attempt, or using a passive list which comprises previously seen nodes that left the overlay unexpectedly. Using a local ring size estimation in combination with a parameter α allows us to define for each node a probability to initiate a merging approach. Each overlay construct initiates on average α merging attempts within a specific time interval, independent of the number of nodes in the overlay construct. We present a simple and a parallelized merging protocol within the Ring Reunion Algorithm. While the simple approach reliably merges multiple Chord rings, the parallelized version systematically initiates further merging attempts at strategic positions in the overlay and accelerates the merging. Evaluation shows that the Ring Reunion Algorithm reliably merges two to five Chord rings in parallel, conducts subsequent mergers quickly, even in complex cases, and leads to a fully correct topology. We identify within the paper ideal values for the fanout parameter and the merge count parameter α for the overlay constructs. The Ring Reunion Algorithm allows ring-based overlays to split and to merge quickly, cost efficiently, and—most of all—reliably. Using ring-based overlays with our merging approach allows us to create data-management applications which can survive network splits on global scale, but also in mobile peer-to-peer networks, and continue their operation once network connection is reestablished.

Author Contributions: Conceptualization, T.A. and K.G.; methodology, T.A. and K.G.; software, T.A. and K.G.; validation, T.A. and K.G.; formal analysis, T.A. and K.G.; investigation, T.A. and K.G.; resources, T.A. and K.G.; data curation, T.A. and K.G.; writing—original draft preparation, T.A.;

writing—review and editing, K.G.; visualization, T.A.; supervision, K.G.; project administration, K.G.; funding acquisition, K.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the Arab German Young Academy of Sciences and Humanities (AGYA).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Dabek, F.; Zhao, B.; Druschel, P.; Kubiatowicz, J.; Stoica, I. Towards a Common API for Structured Peer-to-Peer Overlays. In Proceedings of the International Workshop on Peer-To-Peer Systems, Berkeley, CA, USA, 21–22 February 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 33–44.
2. Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M.F.; Balakrishnan, H. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In Proceedings of the SIGCOMM '01: Proceedings of the International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, San Diego, CA, USA, 27–31 August 2001; ACM: New York, NY, USA, 2001; pp. 149–160. [CrossRef]
3. Rowstron, A.I.T.; Druschel, P. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In Proceedings of the Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Germany, 12–16 November 2001; Guerraoui, R., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2218, pp. 329–350. [CrossRef]
4. Graffi, K.; Masinde, N. LibreSocial: A Peer-To-Peer Framework for Online Social Networks. *Wiley Concurr. Comput. Pract. Exp.* **2021**, *33*, e6150. [CrossRef]
5. Aki, H.M.; Alberro, H.; Firoozi, E.; Qiang, X. Internet Shutdowns—Assessing Their Impact and Effective Countermeasures. Available online: https://www.freiheit.org/publikation/internet-shutdowns (accessed on 26 May 2025).
6. Kis, Z.L.; Szabó, R. Chord-Zip: A Chord-ring Merger Algorithm. *Commun. Lett.* **2008**, *12*, 605–607. [CrossRef]
7. Shafaat, T.M.; Ghodsi, A.; Haridi, S. Dealing with Network Partitions in Structured Overlay Networks. *Peer- Netw. Appl.* **2009**, *2*, 334–347. [CrossRef]
8. Liebau, N.; Pussep, K.; Graffi, K.; Kaune, S.; Beyer, A.; Jahn, E.; Steinmetz, R. The Impact of the P2P Paradigm on the New Media Industries. In Proceedings of the AMCIS 2007: 13th Americas Conference on Information Systems, Keystone, CO, USA, 9–12 August 2007; Association for Information Systems: Atlanta, GA, USA, 2007; p. 255.
9. Kniesburges, S.; Koutsopoulos, A.; Scheideler, C. Re-Chord: A Self-Stabilizing Chord Overlay Network. In Proceedings of the Twenty-Third Annual ACM SPAA: Symposium on Parallelism in Algorithms and Architectures (SPAA'11), New York, NY, USA, 4–6 June 2011; pp. 235–244. [CrossRef]
10. Benter, M.; Soorati, M.D.; Kniesburges, S.; Koutsopoulos, A.; Graffi, K. Ca-Re-Chord: A Churn Resistant Self-Stabilizing Chord Overlay Network. In Proceedings of the 2013 Conference on Networked Systems, NetSys 2013, Stuttgart, Germany, 11–15 March 2013; IEEE Computer Society: Washington, DC, USA, 2013; pp. 27–34. [CrossRef]
11. Datta, A.; Aberer, K. The Challenges of Merging two similar Structured Overlays: A Tale of Two Networks. In *Self-Organizing Systems, First International Workshop, IWSOS 2006, and Third International Workshop on New Trends in Network Architectures and Services, EuroNGI 2006, Passau, Germany, 18–20 September 2006*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4124, pp. 7–22. [CrossRef]
12. Datta, A. Merging ring-structured overlay indices: Toward network-data transparency. *Computing* **2012**, *94*, 783–809. [CrossRef]
13. Aberer, K.; Cudré-Mauroux, P.; Datta, A.; Despotovic, Z.; Hauswirth, M.; Punceva, M.; Schmidt, R. P-Grid: A Self-organizing Structured P2P System. *ACM SIGMOD Rec.* **2003**, *32*, 29–33. [CrossRef]
14. Kis, Z.L.; Szabó, R. Interconnected Chord-rings. *Netw. Protoc. Algorithms* **2010**, *2*, 132–146. [CrossRef]
15. Wu, S.; Raghavan, K.; Di, S.; Chen, Z.; Cappello, F. DGRO: Diameter-Guided Ring Optimization for Integrated Research Infrastructure Membership. *arXiv* **2024**, arXiv:2410.11142.
16. Nguyen, D.; Hoang, N.; Nguyen, B.M.; Tran, V. PSPChord—A Novel Fault Tolerance Approach for P2P Overlay Network. In Proceedings of the Smart Computing and Communication, Tokyo, Japan, 10–12 December 2018; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 386–396. [CrossRef]
17. Kaur, R.; Gabrijelčič, D.; Klobučar, T. Churn Handling Strategies to Support Dependable and Survivable Structured Overlay Networks. *IETE Tech. Rev.* **2022**, *39*, 179–195. [CrossRef]
18. Kaur, R.; Sangal, A.L.; Kumar, K. Churn handling strategies for structured overlay networks: A survey. *Multiagent Grid Syst.* **2017**, *13*, 331–351. [CrossRef]
19. Naik, A.R.; Keshavamurthy, B.N. Next level Peer-to-Peer Overlay Networks under high Churns: A Survey. *Peer-Netw. Appl.* **2020**, *13*, 905–931. [CrossRef]

20. Hussain, A.; Keshavamurthy, B.N. A multi-dimensional routing based approach for efficient communication inside partitioned social networks. *Peer-Netw. Appl.* **2019**, *12*, 830–849. [CrossRef]
21. Amft, T.; Guidi, B.; Graffi, K.; Ricci, L. FRoDO: Friendly Routing over Dunbar-based Overlays. In Proceedings of the IEEE 40th Conference on Local Computer Networks (LCN), Clearwater Beach, FL, USA, 26–29 October 2015; pp. 356–364.
22. Gross, C.; Richerzhagen, B.; Stingl, D.; Münker, C.; Hausheer, D.; Steinmetz, R. Geodemlia: Persistent storage and reliable search for peer-to-peer location-based services. In Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, 9–11 September 2013; IEEE: New York, NY, USA, 2013; pp. 1–2. [CrossRef]
23. Qiu, H.; Ji, T.; Zhao, S.; Chen, X.; Qi, J.; Cui, H.; Wang, S. A Geography-Based P2P Overlay Network for Fast and Robust Blockchain Systems. *IEEE Trans. Serv. Comput.* **2023**, *16*, 1572–1588. [CrossRef]
24. Aquib, M.; Prashanth, P. GeoConnect: Efficient Peer to Peer Network Connectivity for Blockchain Systems. In Proceedings of the 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 24–28 June 2024; IEEE: New York, NY, USA, 2024; pp. 1–6.
25. Fang, J.; Habibi, F.; Bruhwiler, K.; Alshammari, F.; Singh, A.A.; Zhou, Y.; Nawab, F. PeloPartition: Improving Blockchain Resilience to Network Partitioning. In Proceedings of the IEEE International Conference on Blockchain, Blockchain 2022, Espoo, Finland, 22–25 August 2022; IEEE: New York, NY, USA, 2022; pp. 274–281. [CrossRef]
26. Paphitis, A.; Kourtellis, N.; Sirivianos, M. Resilience of Blockchain Overlay Networks. In Proceedings of the Network and System Security—17th International Conference, NSS 2023, Canterbury, UK, 14–16 August 2023; Li, S., Manulis, M., Miyaji, A., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2023; Volume 13983, pp. 93–113. [CrossRef]
27. Marcus, Y.; Heilman, E.; Goldberg, S. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. *IACR Cryptol. ePrint Arch.* **2018**, 236.
28. Saad, M.; Cook, V.; Nguyen, L.N.; Thai, M.T.; Mohaisen, D. Exploring Partitioning Attacks on the Bitcoin Network. *IEEE/ACM Trans. Netw.* **2022**, *30*, 202–214. [CrossRef]
29. Masinde, N.; Graffi, K. Peer-to-Peer-Based Social Networks: A Comprehensive Survey. *SN Comput. Sci.* **2020**, *1*, 299. [CrossRef]
30. Cutillo, L.A.; Molva, R.; Önen, M. Safebook: A distributed privacy preserving Online Social Network. In Proceedings of the 12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WOWMOM 2011, Lucca, Italy, 20–24 June 2011; IEEE Computer Society: Washington, DC, USA, 2011; pp. 1–3. [CrossRef]
31. Guidi, B.; Amft, T.; Salve, A.D.; Graffi, K.; Ricci, L. DiDuSoNet: A P2P architecture for distributed Dunbar-based social networks. *Peer-Netw. Appl.* **2016**, *9*, 1177–1194. [CrossRef]
32. Shafaat, T.M.; Ghodsi, A.; Haridi, S. Managing Network Partitions in Structured P2P Networks. In *Handbook of Peer-to-Peer Networking*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 1127–1147.
33. Graffi, K.; Disterhöft, A. SkyEye: A Tree-Based Peer-To-Peer Monitoring Approach. *Elsevier Pervasive Mob. Comput.* **2017**, *40*, 593–610. [CrossRef]
34. Rapp, V.; Graffi, K. Continuous Gossip-Based Aggregation through Dynamic Information Aging. In Proceedings of the 22nd International Conference on Computer Communication and Networks, ICCCN 2013, Nassau, Bahamas, 30 July–2 August 2013; IEEE: New York, NY, USA, 2013; pp. 1–7. [CrossRef]
35. Binzenhöfer, A.; Staehle, D.; Henjes, R. *Estimating the Size of a Chord Ring*; Technical Report; University of Würzburg: Würzburg, Germany, 2004.
36. Feldotto, M.; Graffi, K. Systematic evaluation of peer-to-peer systems using PeerfactSim. KOM. *Concurr. Comput. Pract. Exp.* **2016**, *28*, 1655–1677. [CrossRef]
37. Feldotto, M.; Graffi, K. Comparative evaluation of peer-to-peer systems using PeerfactSim. KOM. In Proceedings of the International Conference on High Performance Computing & Simulation, HPCS 2013, Helsinki, Finland, 1–5 July 2013; IEEE: New York, NY, USA, 2013; pp. 99–106. [CrossRef]
38. Graffi, K. PeerfactSim.KOM: A P2P System Simulator – Experiences and Lessons Learned. In Proceedings of the IEEE P2P'11: Proceedings of the International Conference on Peer-to-Peer Computing, Kyoto, Japan, 31 August–2 September 2011; IEEE: New York, NY, USA, 2011; pp. 154–155. [CrossRef]
39. Ng, T.S.E.; Zhang, H. Global Network Positioning: A New Approach to Network Distance Prediction. *ACM SIGCOMM Comput. Commun. Rev.* **2002**, *32*, 61. [CrossRef]
40. Matthews, W.; Cottrell, L. The PingER Project: Active Internet performance Monitoring for the HENP Community. *Commun. Mag.* **2000**, *38*, 130–136. [CrossRef]
41. Masinde, N.; Khitman, L.; Dlikman, I.; Graffi, K. Systematic Evaluation of LibreSocial—A Peer-to-Peer Framework for Online Social Networks. *Future Internet* **2020**, *12*, 140. [CrossRef]
42. Wette, P.; Graffi, K. Adding Capacity-Aware Storage Indirection to Homogeneous Distributed Hash Tables. In Proceedings of the 2013 Conference on Networked Systems, NetSys 2013, Stuttgart, Germany, 11–15 March 2013; IEEE Computer Society: Washington, DC, USA, 2013; pp. 35–42. [CrossRef]

43. Disterhöft, A.; Graffi, K. Minicamp: Middleware for Incomplete Participation in Structured Peer-to-Peer Monitoring Protocols. In Proceedings of the 32nd IEEE International Conference on Advanced Information Networking and Applications, AINA 2018, Krakow, Poland, 16–18 May 2018; Barolli, L., Takizawa, M., Enokido, T., Ogiela, M.R., Ogiela, L., Javaid, N., Eds.; IEEE Computer Society: Washington, DC, USA, 2018; pp. 1034–1042. [CrossRef]
44. Ippisch, A.; Sati, S.; Graffi, K. Device to device communication in mobile Delay Tolerant networks. In Proceedings of the 21st IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2017, Rome, Italy, 18–20 October 2017; D'Ambrogio, A., Grande, R.E.D., Garro, A., Tundis, A., Eds.; IEEE Computer Society: Washington, DC, USA, 2017; pp. 91–98. [CrossRef]
45. Ippisch, A.; Graffi, K. Infrastructure Mode Based Opportunistic Networks on Android Devices. In Proceedings of the 31st IEEE International Conference on Advanced Information Networking and Applications, AINA 2017, Taipei, Taiwan, 27–29 March 2017; Barolli, L., Takizawa, M., Enokido, T., Hsu, H., Lin, C., Eds.; IEEE Computer Society: Washington, DC, USA, 2017; pp. 454–461. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.