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

Tobias Amft Kalman Graffi

Heinrich Heine University, Düsseldorf, Germany
Computer Science Department
Technology of Social Networks Group

TECHNICAL REPORT TR-2017-001
HEINRICH HEINE UNIVERSITY, DÜSSELDORF, GERMANY
COMPUTER SCIENCE DEPARTMENT

SEPTEMBER 2017

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

Tobias Amft and Kalman Graffi
Technology of Social Networks Group, Heinrich-Heine University Düsseldorf, Germany
Email: [amft, graffi]@cs.uni-duesseldorf.de

***Abstract*—Peer-to-peer overlays define an approach to operate 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 it 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 Ring Reunion Algorithm and point out how to reduce the amount of messages both in separated and united overlay states. The algorithm is parallelized for an accelerated merging and able to automatically detect overlay partitioning and to start corresponding merging processes. We evaluated through simulations the new Ring Reunion Algorithm in its simple and parallelized form in comparison to plain Chord, Chord-Zip and two versions of the Ring Unification Algorithm. Evaluation shows that only our parallelized Ring Reunion Algorithm allows to merge 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.**

I. Introduction

Speech is the most outstanding ability of mankind. It contributes to communication, sympathy and to disclosure of information among people all over the world. Unfortunately, still many countries exist in which freedom of speech is suppressed by the respective government. According to the *Amnesty International Report 2013* the right to speak freely was suppressed in 101 countries in 2012 [2]. The important role of media in the context of information dissemination cannot be denied. Probably the most important medium nowadays is the Internet, which provides the ability to globally access information. Therefore it is not surprising that governments of many countries attempt to control parts of the Internet to control its citizens likewise. Control is conducted twofold: first, specific services or websites in the Internet can be shut down and second, if needed, the entire connectivity of the country to the world can be shut down.

One drastic and current example of suppressing freedom of speech and the dissemination of information is the behavior of the Egypt government during the so-called Arab Spring in early 2011. The uniqueness of the protests during the Arab Spring is characterized by the enormous usage of social media and familiar Internet platforms like Facebook, Twitter and YouTube, which mainly have been used to organize demonstrations, to share informational content or simply to criticize the government. As a response to the revolutionary movements in North Africa, the Egyptian government instructed mobile operators and Internet service providers to suspend their services. As a result, from 27^{th} of January 2011 until 2^{nd} of February 2011 approximately 93% of Egyptian networks had been unreachable [6], most parts of the Internet in Egypt had been cut off by the Egyptian government. Only few governments had done this before: Nepal cut off Internet access entirely in 2005, as did Myanmar two years later in 2007 [15].

The cases of Nepal, Myanmar or Egypt show that governments of countries with simple Internet infrastructure are capable of stopping transnational Internet traffic almost entirely. In such a case most centralized Internet services, e.g. social media, are not reachable, as the provider's servers are mostly located abroad.

Another problem arises with traditional client-server approaches: centralized servers are convenient to intercept and manipulate information directly. The findings of Edward Snowden on surveillance in the Internet are a current evidence that Internet services like Facebook, Yahoo, Twitter and so forth are well suited for being spied on. Almost all information about users of these websites are gathered centrally at the provider's servers and can be accessed there by interested agencies.

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 the peers in a decentralized manner without the need of a dedicated server that could be attacked. The peer-to-peer architecture is therefore suitable for sharing data or information dissemination. Nevertheless, as history shows, it is important for peer-to-peer solutions to be aware of network partitions, as they might occur unexpectedly at any time, whether due to imperious governments or because of natural catastrophes, which might disrupt parts of the Internet, as an earthquake did in Taiwan in December 2006 [3].

In case a network partitioning event occurred, participants of a disrupted peer-to-peer overlay should be capable of maintaining the network in the corresponding geographical regions. The split of the overlay should be survived and the overlay network should quickly stabilize. In addition, and

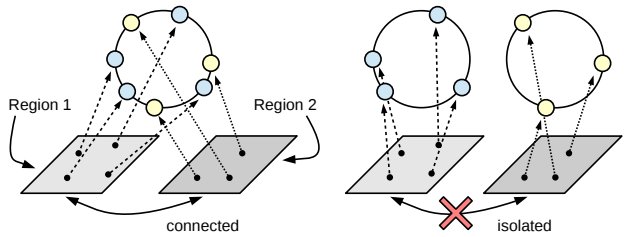


Fig. 1. Ring is corrupted during isolation.

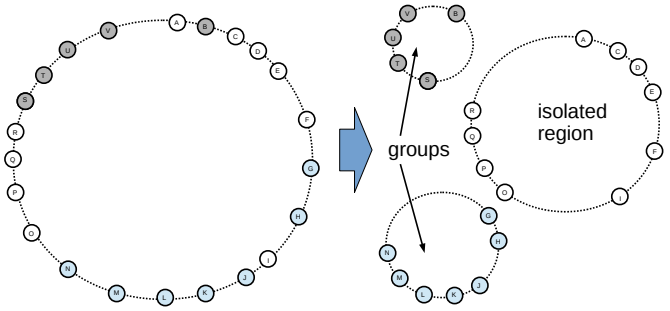


Fig. 2. Groups are formed after one region has been isolated.

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

In the last decade, different types of overlays have 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 [20] as example for this category of overlays, as it is the most cited structured peer-to-peer overlay. It allows detailed investigations on this issues, as solutions may be transferable to other ring-based overlay networks.

In this paper, we analyze existing overlay merging algorithms and present a novel merging algorithm named Ring Reunion Algorithm, aiming at quick stabilization, and low costs for the detection and merging of partitioned overlays.

A. Characteristics of Network Partitioning

As we know, autonomous systems (*AS*) in the Internet are connected by Internet exchange points (IXP) and controlled by different Internet service providers (*ISP*). 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 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 sudden separation of whole groups is rarely considered. This happens when a set of nodes, which is associated to 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 occur, 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. 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.

B. 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. In specific, 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 II 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 III and evaluate in Section IV our proposed approach in comparison to Chord-Zip [12], the simple and Gossip-based Ring Unification Algorithm [18] and the unmodified stabilization algorithms of Chord.

II. Related Work

While research on peer-to-peer overlays has become popular in the last decade, the issue of maintaining such overlays in the presence of network partitioning has been discussed only in

few works. The authors of Re-Chord [14] and the churn-aware extension Ca-Re-Chord [4] present a self-stabilizing protocol which recovers a Chord ring from any initial state in which nodes are weakly connected. Earlier in 2005, Shaker et al. [19] proposed a protocol to let peers, which are connected in an arbitrary state, to converge to a structured peer-to-peer overlay based on a ring structure. In this paper we focus on recovery from partitioning events during which nodes might not even be connected to great parts of the overlay.

Merging algorithms are used to stabilize overlays during network partitioning events by detecting and uniting participants became separated from the overlay. As fist step, a merging algorithm should be able to detect reachable nodes which have been separated. Especially nodes which belong to the same region 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. Below, we summarize different merging algorithms from literature and compare them to our Ring Reunion Algorithm in Section IV.

A. Tale of Two Networks

Datta et al. characterize in [8] challenges of merging two structured overlays, which base on the same protocol. The authors compare ring-based overlays (Chord [20]) with overlays which use prefix-based routing and structural replication (P-Grid [1]). Later in [7], Datta investigates Chord's performance during the process of merging two Chord rings by summarizing the robustness of routing in Chord during a merger.

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 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.

B. Chord-Zip

The authors of Chord-Zip [12], [13] 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 case 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.

```
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
```

Fig. 3. Chord Zip Algorithm.

The execution time of the merger, which consists of one Chord join operation and the merge signal, can be improved by determining 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 to appoint 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 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. 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.

C. Ring Unification Algorithm

Shafaat et al. present in [18] an algorithm to merge similar structured, unidirectional ring-based overlays like Chord. First, they present a simple version of their Ring Unification Algorithm to demonstrate their idea of a merging algorithm. In addition, they present an advanced version which uses a parameter, termed fanout, to adjust the trade-off between message complexity and time complexity. More specific, the fanout parameter regulates the number of merger processes which are run in parallel. 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 the two version of the Ring Unification Algorithm in the following.

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 as well as 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 to identify potential counterparts 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 examines if the information about the newly found nodes fits better into the routing table. If so, a node updates it's routing table and informs neighboring nodes about possible merging candidates. As a result the merger proceeds in both directions, clockwise and anti-clockwise.

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. Goal of the enhancement is 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 that too many messages are produced. After evaluating the algorithm with different parameters, the authors suggest a fanout value around 3-4 as a good trade-off between message overhead and time complexity.

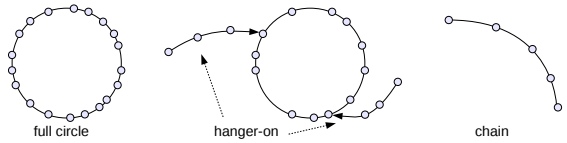


Fig. 4. Different constructs during merging process.

D. Conclusions on Related Work

Only few work in literature explicitly addresses the issues of splitting and merging peer-to-peer overlays. The approach presented by Datta et al. in [8] 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 to Chord, but only a single merger process traversing the overlay. We expect that this approach is both, slow, as well as sensitive to churn, the merger process might get lost and the merging fails, 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 is bound to the passive list, the set of potentially mergeable overlays is limited. Also here we expect and show in the evaluation that the merging process can fail.

III. 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 IV-A) in the network and not to the number of nodes. As constructs we define patterns that form if nodes are interconnected by its successor pointers. We distinguish between three different constructs: full circles, hanger-ons and chains (see Figure 4), regarding on 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.

For the presentation of our approach, we first describe in Subsection III-A how a single node can identify potential other nodes in other overlays. Once a potential node is identified, we discuss in Subsection III-B how the nodes in an overlay construct coordinate to limit the number of merging attempts per overlay construct. With a potential contact at hand, we present in Subsection III-C our merging protocol and point

out in Subsection III-D how the merging process terminates. In Subsection III-E we extend our Ring Reunion Algorithm to initiate coordinated merge processes in parallel. This approach, termed in the evaluation *Reunion2,3,4*, improves the merging performance and robustness. Finally, in Subsection III-F we discuss how our Ring Reunion Algorithm handles the parallel merging of multiple overlay rings.

A. Discovery of Separated Overlays

For the merging of several unconnected overlays, at first it is necessary that at least one node in one overlay (the initiator) gets in contact with a node from an other overlay (contact node) to start a merging algorithm. Like in the Ring Unification Algorithm [18], as first option a *passive list* can be used here to start new, independent instances of the Ring Reunion Algorithm. Nodes which are detected to be offline are moved from the routing table into the passive list, where they are periodically tested for reachability. Nodes in the passive list which 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 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.

B. 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. In specific, 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 α defines the number of started merging operations. Thus, within a time frame any separated overlay aims at finding new contacts and initiating

```
1  receipt of STARTMERGER(contact) from m at n
2     sendto contact : LOOKUP (n.id)
3     if altSucc is received from contact within time interval
            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
```

Fig. 5. Ring Reunion Algorithm.

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. Optimistically, a good estimation for the size of of the network is also sufficient. Thus we follow an approach similar to [5] 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 these 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.

C. Merging Algorithm

If a contact node is found and the probability condition is fulfilled, 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, Fig. 5), or by sending the contact details of the alternative successor to its successor (if $altSucc \notin (n,\ succ)$, line 13, Fig. 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

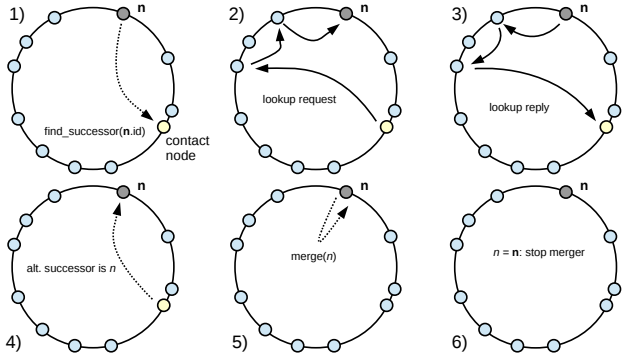


Fig. 6. Node tries to merge own ring. Merger stops immediately when node receives itself as contact.

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, node for node and successor for successor, the two overlay constructs are merged.

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 $O(N+logN)$, where N denotes the number of nodes participating in the overlay.

D. Termination

The Ring Reunion Algorithm terminates if the received alternative successor is equal to the node which holds the merger token (line 9, Fig. 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) already has merged the ring locally.

```
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
```

Fig. 7. Ring Reunion Algorithm with extension to start parallel instances of the merging algorithm.

E. 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.

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 Fig. 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. Thereupon those finger contacts inform their second furthermost finger contacts respectively et cetera, that thereby the information to start new instances is equally distributed among all nodes in the initiator node's ring (lines 7-10). 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 (see line 7 in Figure 7).

Hereafter, all nodes receiving 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.

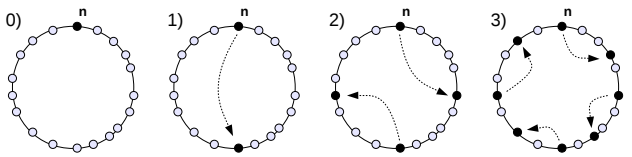


Fig. 8. Distribution algorithm: furthermost finger contacts are invited to start instances of the merging algorithm as well. These nodes are informed about the presence of a second overlay construct and start a merging attempt in this second overlay upon receiving related contact information.

The terms *Reunion(i)*, e.g. Reunion2, Reunion3, etc., in our evaluation correlate to the exponent i.

F. Example of Merging of several Rings

As the following example 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 directly after node 18's MERGE message has arrived at node 27. Arrows describe forwarded MERGE messages, solid lines denote the successor pointers from each node to its successor.

IV. Evaluation

In this section we evaluate the behavior of our proposed Ring Reunion Algorithm, both in the unparallelized (*Reunion*) and parallelized version (*Reunion2,3,4*) in comparison to Chord-Zip [12] (*Chord-Zip*) and Shafaat et al.'s [18] simple Ring Unification (*Simple*) and Gossip-based Ring Unification (*Gossip*) Algorithms.

A. 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, that means multiple constructs should be combined in the 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. In the following, we discuss the metrics in detail.

a) 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. As constructs we define patterns that form in the overlay during a network partitioning event. An overview of the constructs we define is given in Section III, Figure 4. 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.

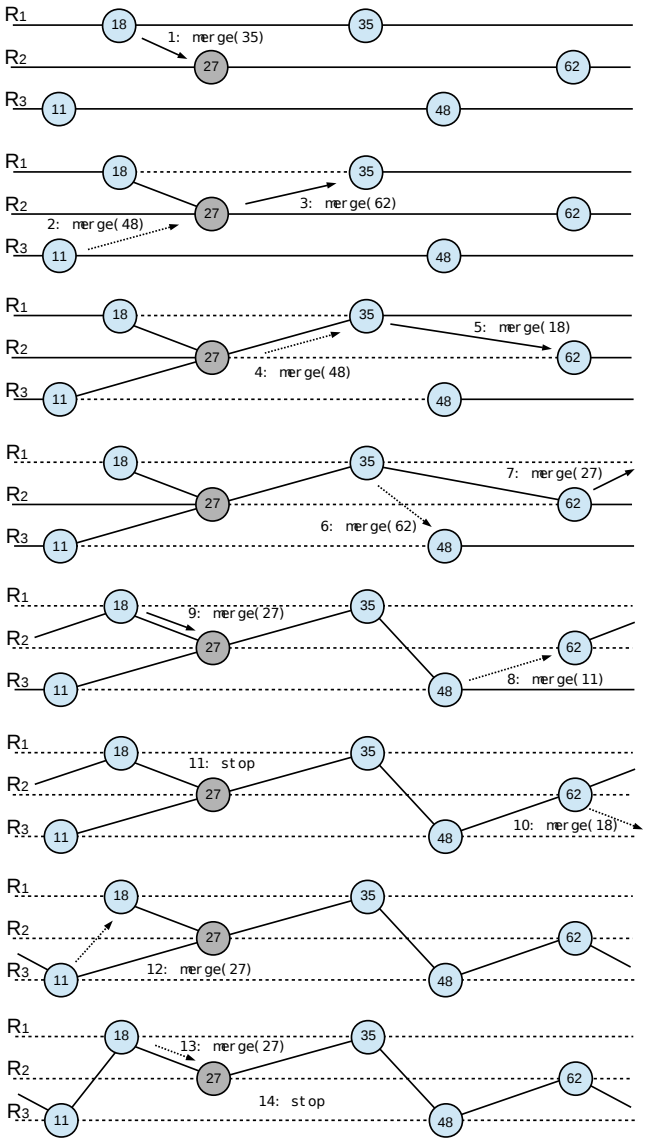


Fig. 9. Example of two nodes merging the same ring, using the Ring Reunion Algorithm.

b) 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.

c) 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 a merging approach should be in relation to the number of constructs in the underlying network. With one large ring, the overhead should be minimal, while with more rings and constructs the contact searching and merging overhead is acceptable to rise.

General Settings	
Simulator details	PeerfactSim.KOM [10] [9] [11], 10 seeds per setup
Network model	GNP [17], jitter based on [16], no packet loss
A) Simultaneous merging of three networks	
Algorithms	Chord-Zip, Simple Ring Unification, Gossip-based Ring Unification, 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 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 10242 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, Simple Ring Unification, Gossip-based Ring Unification, 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 minutes. If active node is found, start merging process.
C.2 Public list	Publicly known 160 contacts are tested every 4 minutes 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 minute. 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 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 minutes.
E.2 Changing frequency of merge attempts	Algorithm combined with intervals of 5,10,15 and 20 minutes in which merger instances are started, to find compromise between operation time and bandwidth consumption.

TABLE I
SIMULATOR SETUP AND DIFFERENT SCENARIO SETUPS.

B. Simulation Setup

We use PeerfactSim.KOM [10] [9] [11], an event-based simulator for peer-to-peer protocols, for our simulations, 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 of 10 different values. Each algorithm has been tested separately and independently from other merging algorithms. In addition, each simulation uses GNP coordinates [17] to estimate delays in the fundamental network realistically. The network layer was extended to strictly isolated dedicated regions, any connection in or out selected regions, is dropped in selected times. Reasonable approximations for jitter and message delay are integrated into the simulator by using measurements from the PingEr project [16]. In most of our simulations 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 I. First, with Setup A.1 we investigate the performance of various merging algorithms while merging three overlay networks with 1024 nodes in total without the presence of network partitioning events. This scenario should reveal if each algorithm is able to unify multiple separated Chord rings. We want to filter bogus mechanisms out and obtain a first feeling 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 10^{th} 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 minutes the simulation is finished.

Next, in Setup B.1 we compare the performance of the most promising approaches from A.1 in a large-scale network with 10242 nodes. Two Chord rings are formed a the beginning of the scenario. In the 10^{th} 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 minutes. The Gossip Based Ring Unification Algorithm is tested with a fanout parameter of 4, since this value is suggested by the algorithms authors in [18] This setup finally shows that our Ring Reunion Algorithm and its parallelized versions *Reunion2,3,4 (4,8,16 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 various approaches to identify new nodes for merging in the network. The simulations of Setups C.1-2 begin with a join phase. After 150 minutes 1024 nodes have build a global Chord ring. In the 180^{th} 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 minutes 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 below 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 1, 5, 10 and 100 for α. Similar to the previous scenario a Chord ring with 1024 participants is formed from minute 0 to minute 150. After 180 minutes a group of 400 nodes is isolated, after 240 minutes it is connected again to the remaining groups and after 360 minutes 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 4 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 minutes. A group of 400 nodes is then isolated from the 180^{th} minute to the 240^{th} minute. In addition, another group of 50 nodes is isolated from the 200^{th} minute to the 240^{th} minute. Finally, from minute 240 to 300 a third group of 100 nodes is isolated from the other regions.

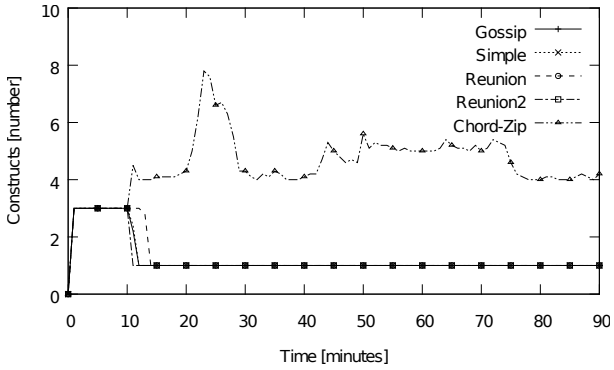
In the previous scenarios we did not consider churn in order to focus on the fundamental behavior of each tested algorithm. Finlly, in Setups E.1-2, a group of 400 nodes is separated from the rest of the Chord ring in minute 180, when all nodes have joined the network successfully. The duration of the isolation lasts 60 minutes. From minute 240 to 360, when the simulation finishes, the nodes unify the partitioned network again. 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 under presence of churn. Supplementary parameters we examine in this scenario are the parameter which regulates the number of parallelized merger instances and the interval within which the *public list* is iterated.

V. Evaluation Results

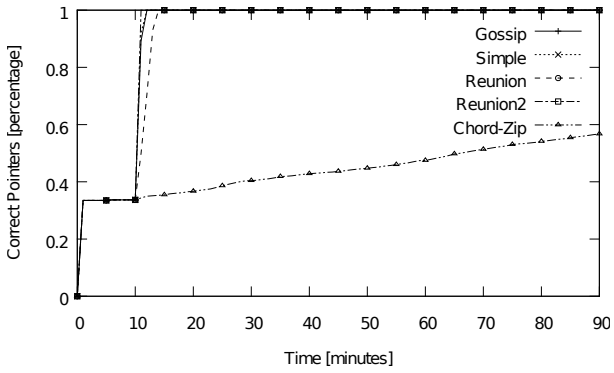
In this section, we present the simulation results which have been obtained from simulating the scenarios mentioned before. First, in Section V-A, we test the basic functionality of Chord-Zip, the simple and Gossip-based Ring Unification Algorithm and our Ring Reunion Algorithm by merging three Chord rings simultaneously. In V-B we directly compare the performance of our Ring Reunion Algorithm with the Gossip-based Ring Unification Algorithm. Section V-C 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 V-D. To conclude our studies, we investigate our Ring Reunion Algorithm in the presence of churn in Section V-E.

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

In Setup A.1, three different Chord rings, with 341, 341 and 342 nodes per ring, are created. 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 is



(a) A.1 Three Networks, Constructs.

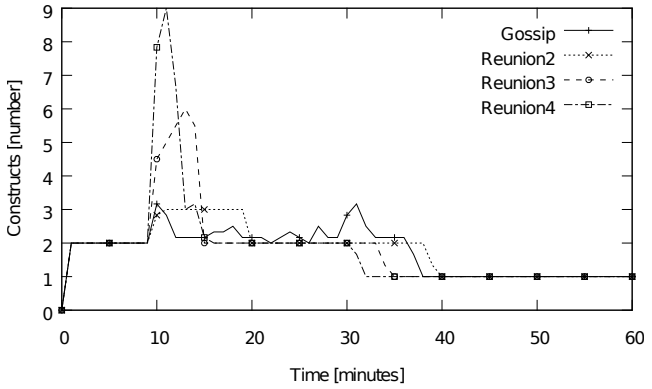


(b) A.1 Three Networks, Correct Pointers.

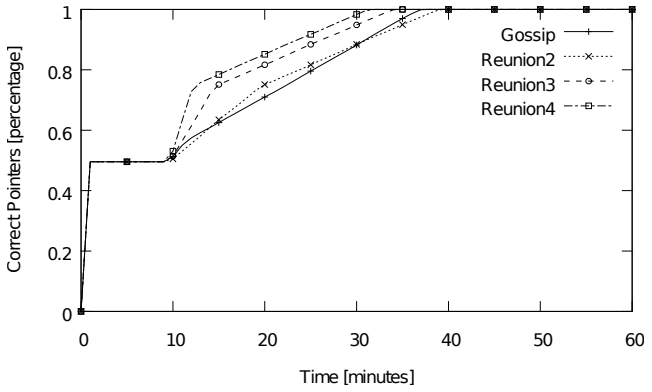
Fig. 10. A.1 Merging of Three Networks.

able to 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 10(b) one can see that the Chord-Zip Algorithm needs more time to adjust the successor pointers than the other merging algorithms. Figure 10(b) 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 10(b) 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) B.1 Constructs.



(b) B.1 Correct Pointers.

Fig. 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.

B. 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 way 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 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 counter 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 to the number of messages with a small number of instances.

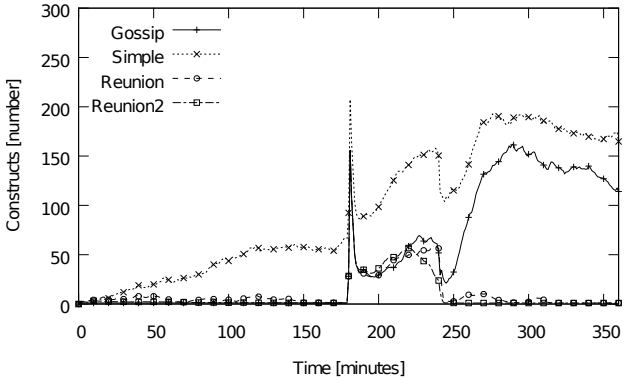
C. Evaluation Results for Setups C.1-5: Testing Various Approaches to Start Merging Instances

The scenario of Setups C.1-5 is the following: 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 separated nodes again. In Setups C.3-5 we reduce overhead in terms of message complexity by introducing a probabilistic model to start merging procedures.

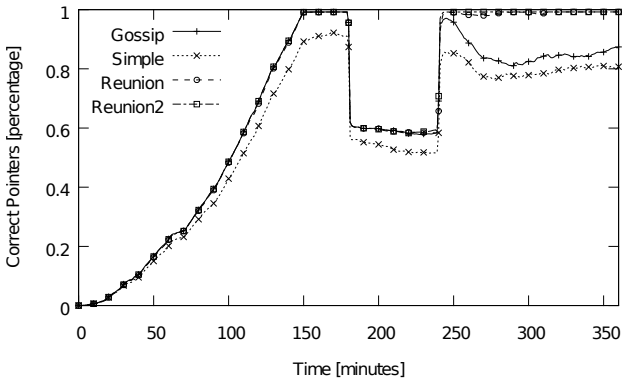
In Figures 12(a), 12(b), and 12(c) the usage of a passive list is shown. Nodes which are detected by a node to be suddenly unavailable are added to its passive list which is iterated periodically to find possible contact nodes to merge. Observing the large number of constructs for Chord-Zip, one can see 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 12(a). Only the Ring Reunion Algorithm has been able to adjust all successor pointers directly after the partitioning event finished at minute 240 (Figure 12(b)). Figure 12(c) 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.

As Figure 13(b) indicates, only the Ring Reunion Algorithm manages to adjust all successor pointers in the simulated time. Again the Gossip-based Ring Unification Algorithm forms a large amount of constructs after all network partitions have become reachable again, see Figure 13(a). 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. Figures 13(a) and 13(c) show that the message consumption of both algorithms is highly related to the number of constructs in the underlying network.

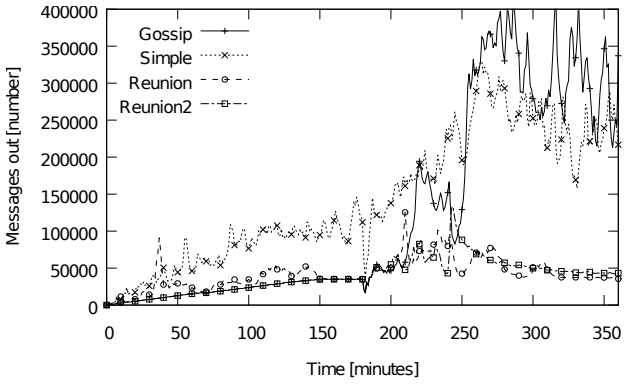
At first sight it might seem questionable why the public list, which actively repeats to contact the same overlay nodes again and again, should be considered as alternative to the passive list (reactive approach). The answer is very simple though: the problem with the passive list is that only nodes 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



(a) C.1 Passive List: Constructs.

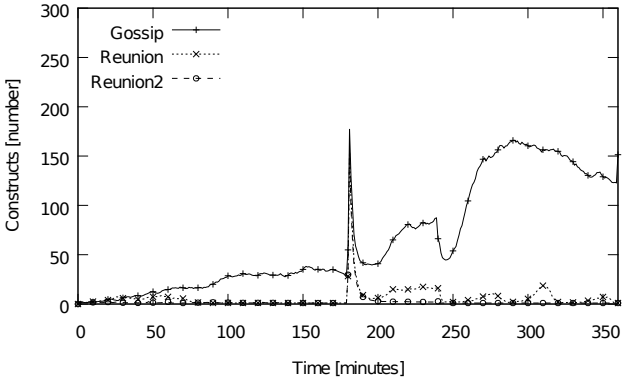


(b) C.1 Passive List: Correct Pointers.

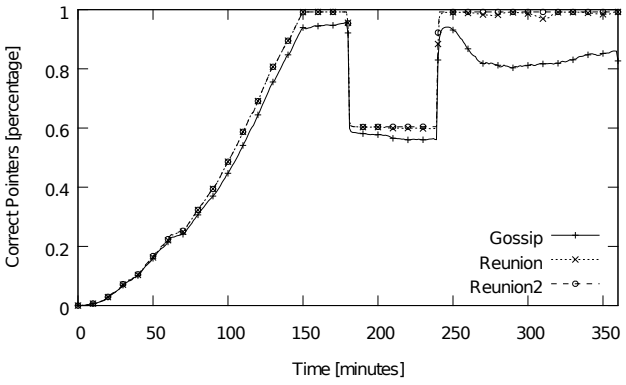


(c) C.1 Passive List: Message Overhead.

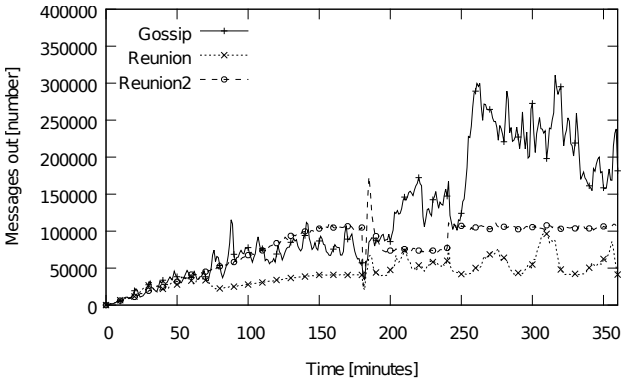
Fig. 12. C.1 Nodes maintain passive list to find further overlays in case of network partitioning event. Merging algorithm is started automatically if contact node is reachable.



(a) C.2 Public List: Constructs.



(b) C.2 Public List: Correct Pointers.



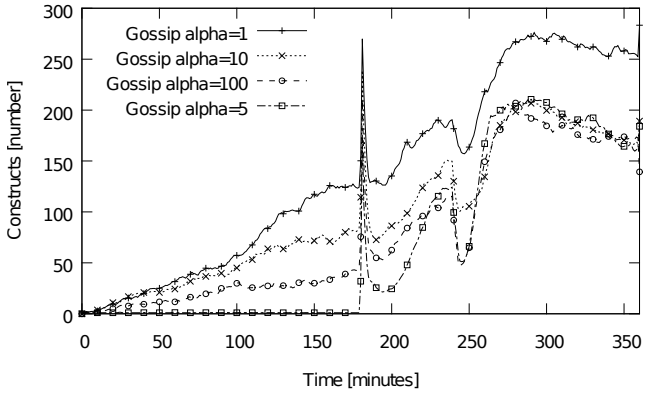
(c) C.2 Public List: Message Overhead.

Fig. 13. C.2 Public list is iterated to find further overlays. If contact node on public list is reachable, a merger instance is initiated.

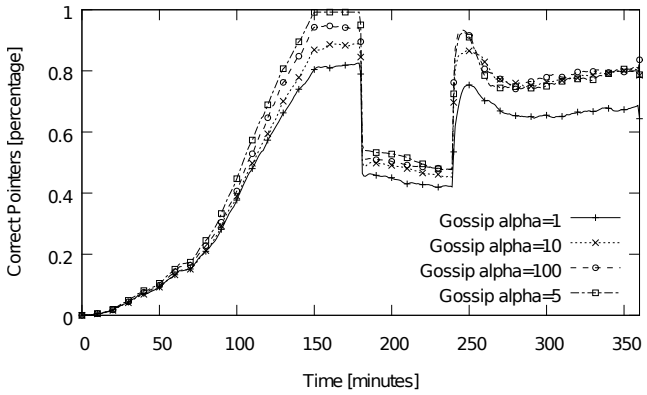
which are not dependent on the routing table.

The principle of the public list which contains randomly selected nodes counteracts the effect of unintended group forming, stated in Section I-A, 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-time 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 simulation in order to simulate a worst case scenario in which no bootstrap server is available once a node joined the network. A comparison of Figures 12(c) and 13(c) shows that the message overhead of the public list (active approach) is comparable to the overhead produced by the passive list (reactive approach).

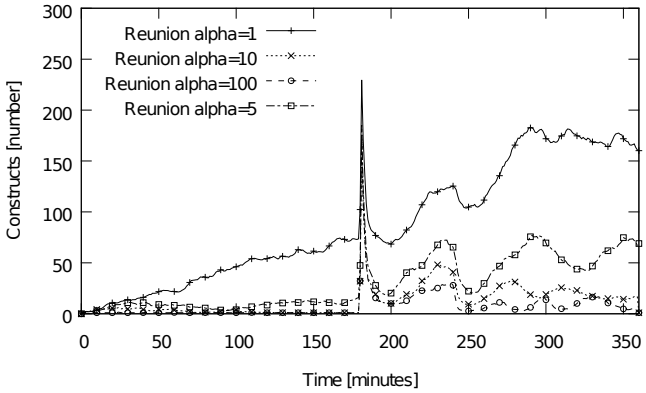


(a) C.3 Public List, Gossip: Constructs.

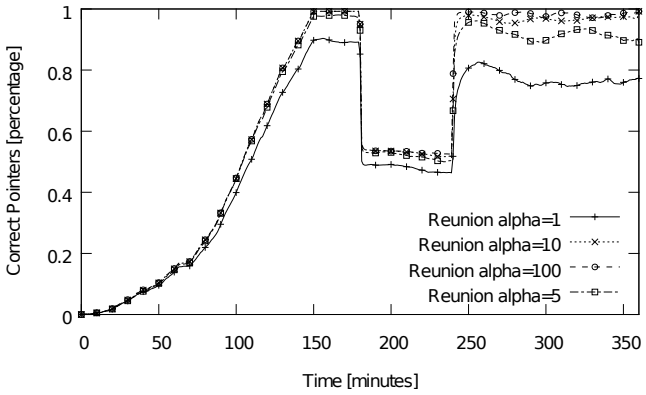


(b) C.3 Public List, Gossip: Correct Pointers.

Fig. 14. C.3 Parameter study: Gossip-based Ring Unification Algorithm (8 instances). Each overlay construct initiates α mergers within a time period of 4 minutes.



(a) C.4 Public List, Reunion: Constructs.



(b) C.4 Public List, Reunion: Correct Pointers.

Fig. 15. C.4 Parameter study: Ring Reunion Algorithm. Each overlay construct initiates α mergers within a time period of 4 minutes.

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.

This setups and corresponding results in Figure 14 allow us to determine a good value for α, i.e. the number of merger instances per construct. As can be seen in Figure 14(a) the Ring Unification Algorithm performs the poorer the less merger instances are started. Considering Figure 16(b) we see that our Ring Reunion Algorithm performs well if approximately 10 instances per construct are started in combination with 4 initiated mergers in parallel. Furthermore, we learn from this evaluation result that it is necessary to react quickly on partitioning events to perform well and to reduce costs.

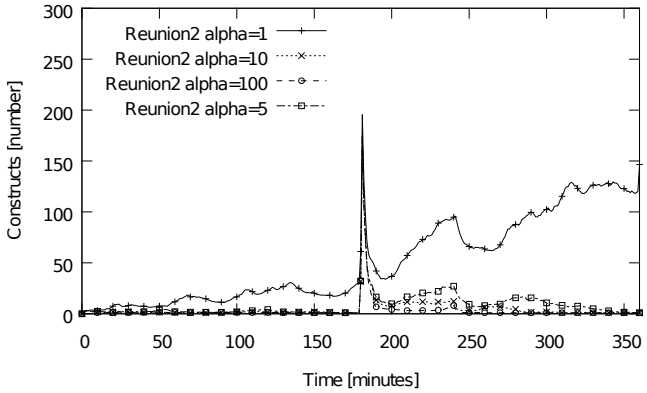
D. Evaluation Results for Setup D.1: A Complex Scenario

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

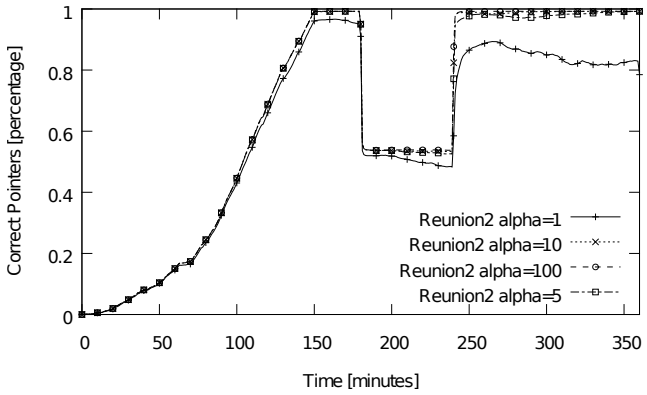
Figure 17(a) verifies our expectation: the Ring Reunion Algorithm with 4 parallel instances is able to merge all reachable regions fast enough to handle even multiple network failures. Figure 17(a) shows that our Ring Reunion Algorithm corrects faulty pointers quickly after a partitioning event at minutes 180, 200 and 240. In addition one can obtain from Figure 17(b) 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 low traffic overhead.

E. 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

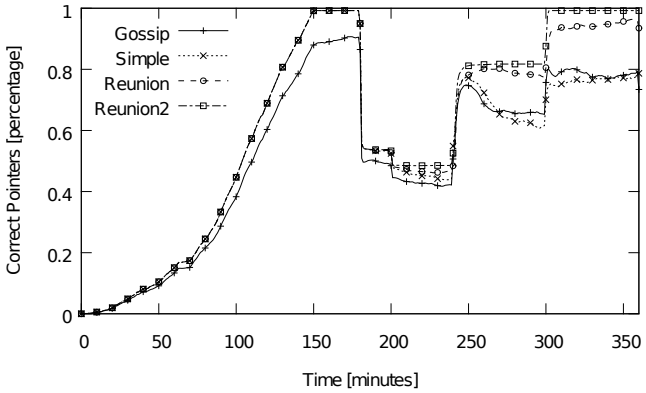


(a) C.5 Public List, Reunion2: Constructs.

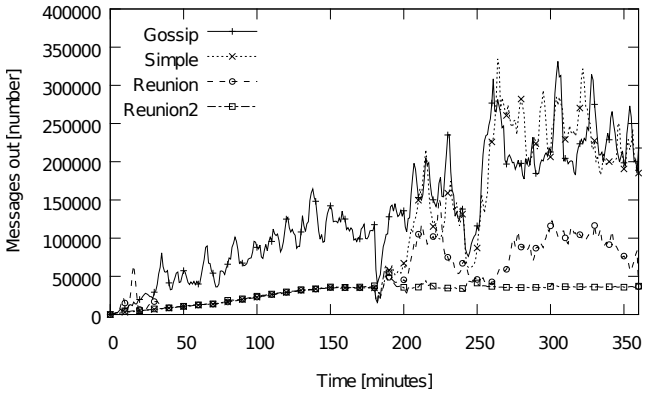


(b) C.5 Public List,Reunion2: Correct Pointers.

Fig. 16. C.5 Parameter study: parallelized Ring Reunion Algorithm (4 instances). Each overlay construct initiates α mergers within a time period of 4 minutes.



(a) D.1 Correct Pointers.



(b) D.1 Message Overhead.

Fig. 17. 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.

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 specific, 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 minutes. 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 minutes to 20 minutes in steps of 5 minutes. In both setups, we investigate the effect of parallel merger instances on the Ring Reunion Algorithm reliability.

Figure 19 reveals the behavior of the Ring Reunion Algorithm if multiple parallel instances are distributed by the first initiator node. In this scenario, every 5 minutes the *public list* tests a contact node to be reachable. In Figures 18(a) and 18(b) 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 18(d). Nevertheless, Figures 19(a), 19(b), 19(c) and 19(d) 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 21 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 21, it can be observed that high numbers of parallel instances operate the better, the less attempts are 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 in addition multiple merger instances are 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

20(d) for all intervals greater than 5 minutes. Furthermore, in some cases, the determination of the additional merger instances takes longer time than the actual merger process or the interval within which new merger attempts are started. Hence, the additional started merger instances tear up the current overlay constructs again. In 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 are 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.

To conclude our studies, we suggest 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 and too often started merging processes can cause an opposite effect of the Ring Reunion Algorithm. Configured properly instead, the Ring Reunion Algorithm is able to operate fast and reliably without producing to much overlay constructs.

VI. Conclusions

In this paper, we address the issue of peer-to-peer overlays in the presence of dynamic Internet partitioning on national scale. Merging algorithms promise to unify separated overlays after such partitioning events. We show that Chord [20] as well as previously proposed merging algorithms for ring-based overlays such as Chord-Zip [12] and the Ring Unification Algorithm [18] are incapable to reliably merge 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 to merge overlays after heavy network partitioning events.

We present in this paper a novel merging algorithm for ring-based peer-to-peer overlays, named 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 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 merges reliably multiple Chord rings, the parallelized version systematically initiates further merging attempts at strategic positions in the overlay and accelerates the merging.

Our 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 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.

References

[1] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-Grid: A Self-organizing Structured P2P System," *ACM SIGMOD Record*, vol. 32, no. 3, pp. 29–33, 2003.

[2] Amnesty International, "Amnesty International Report 2013," https://www.amnesty.org/en/annual-report/2013/downloads, 2013. Accessed: Feb. 2016.

[3] BBC News, "Asia Communications hit by Quake," http://news.bbc.co.uk/2/hi/asia-pacific/6211451.stm, 2006. Accessed: Feb. 2016.

[4] M. Benter, M. Divband, S. Kniesburges, A. Koutsopoulos, and K. Graffi, "Ca-Re-Chord: A Churn Resistant Self-stabilizing Chord Overlay Network," in *Proc. of Networked Systems Annual Technical Conf. (NetSys)*, 2013.

[5] A. Binzenhöfer, D. Staehle, and R. Henjes, "Estimating the Size of a Chord Ring," in *University of Würzburg, Technical Report*, 2004.

[6] CircleID, "Egyptian Government Shuts Down Most Internet and Cell Services," http://www.circleid.com/posts/egyptian_government_-shuts_down_most_internet_and_cell_services/, 2011. Accessed: Feb. 2016.

[7] A. Datta, "Merging ring-structured overlay indices: toward network-data transparency," *Computing*, vol. 94, no. 8-10, pp. 783–809, 2012.

[8] A. Datta and K. Aberer, "The Challenges of Merging two similar Structured Overlays: A Tale of Two Networks," in *Proc. of Int. Conf. on Self-Organizing Systems (SOS)*, 2006.

[9] M. Feldotto and K. Graffi, "Comparative evaluation of peer-to-peer systems using peerfactsim. kom," in *Proc. of Int. Conf. on High Performance Computing and Simulation (HPCS)*, 2013.

[10] ——, "Systematic evaluation of peer-to-peer systems using peerfactsim. kom," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 5, pp. 1655–1677, 2015.

[11] K. Graffi, "PeerfactSim.KOM: A P2P System Simulator – Experiences and Lessons Learned," in *Proc. of Int. Conf. on Peer-to-Peer Computing (P2P)*, 2011.

[12] Z. L. Kis and R. Szabó, "Chord-Zip: A Chord-ring Merger Algorithm," *Communications Letters*, vol. 12, no. 8, pp. 605–607, 2008.

[13] ——, "Interconnected chord-rings," *Network Protocols and Algorithms*, vol. 2, no. 2, pp. 132–146, 2010.

[14] S. Kniesburges, A. Koutsopoulos, and C. Scheideler, "Re-chord: a self-stabilizing chord overlay network," *Theory of Computing Systems*, vol. 55, no. 3, pp. 591–612, 2014.

[15] Matt Richtel, "Egypt Cuts Off Most Internet and Cell Service," http://www.nytimes.com/2011/01/29/technology/inter-net/29cutoff.html, 2011. Accessed: Feb. 2016.

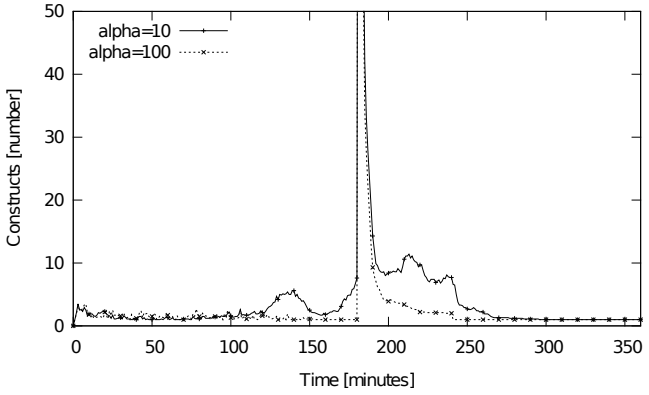
[16] W. Matthews and L. Cottrell, "The PingER Project: Active Internet performance Monitoring for the HENP Community," *Communications Magazine*, vol. 38, no. 5, pp. 130–136, 2000.

[17] T. S. E. Ng and H. Zhang, "Global Network Positioning: A New Approach to Network Distance Prediction," *ACM SIGCOMM Computer Communications Review*, vol. 32, no. 1, pp. 61–61, 2002.

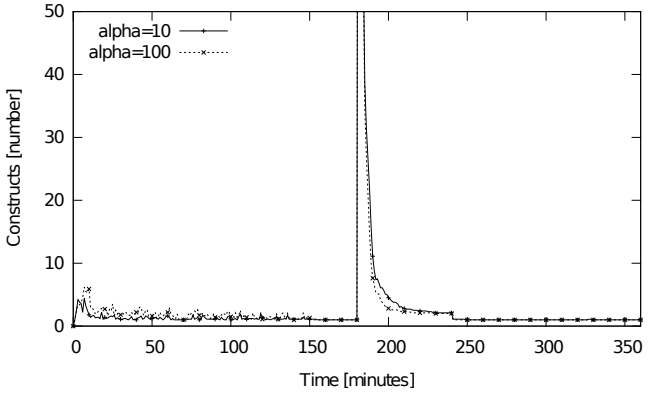
[18] T. M. Shafaat, A. Ghodsi, and S. Haridi, "Dealing with Network Partitions in Structured Overlay Networks," *Peer-to-Peer Networking and Applications*, vol. 2, no. 4, pp. 334–347, 2009.

[19] A. Shaker and D. S. Reeves, "Self-stabilizing structured ring topology p2p systems," in *Proc. of Int. Conf. on Peer-to-Peer Computing (P2P)*, 2005.

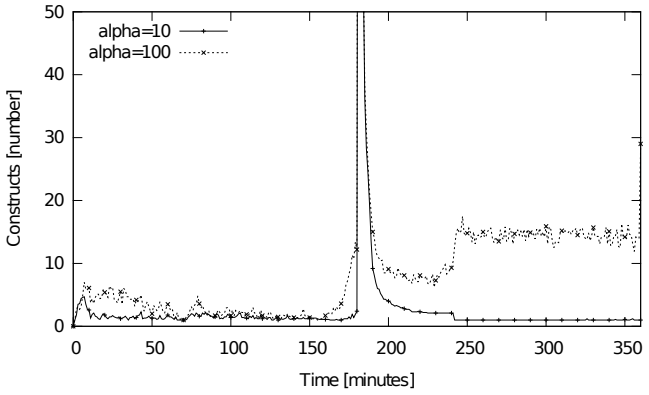
[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. of Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2001.



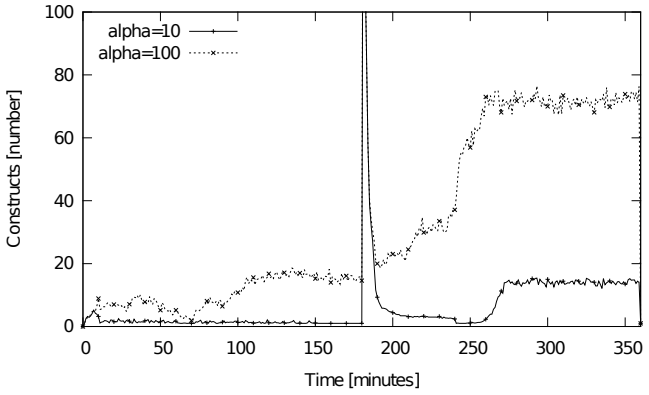
(a) E.1: 8 Instances, Constructs.



(b) E.1: 16 Instances, Constructs.

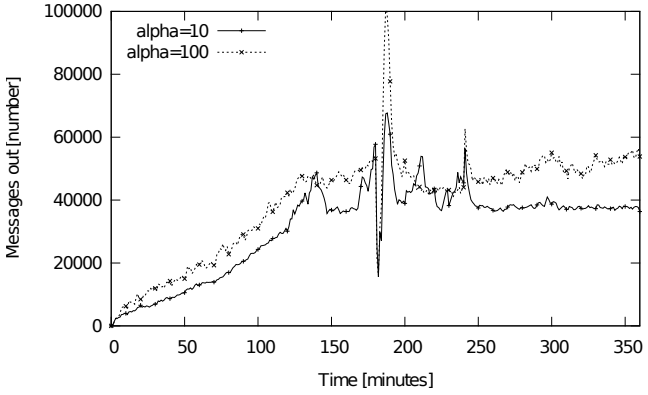


(c) E.1: 32 Instances, Constructs.

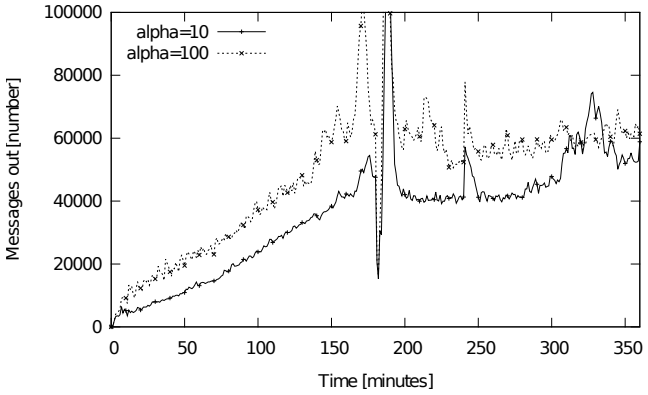


(d) E.1: 64 Instances, Constructs.

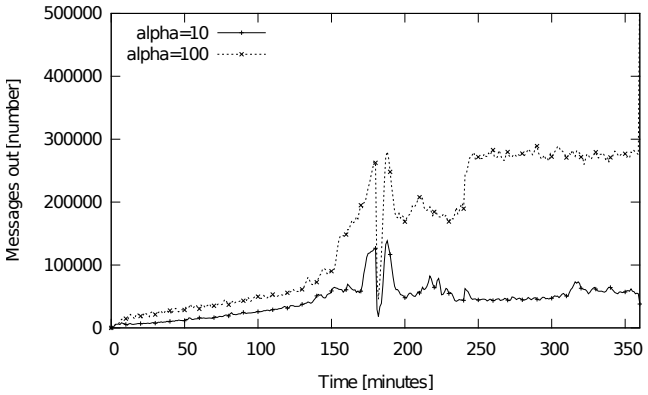
Fig. 18. E.1: Ring Reunion Algorithm with parallel instances, $\alpha \in \{10, 100\}$, constructs during merger.



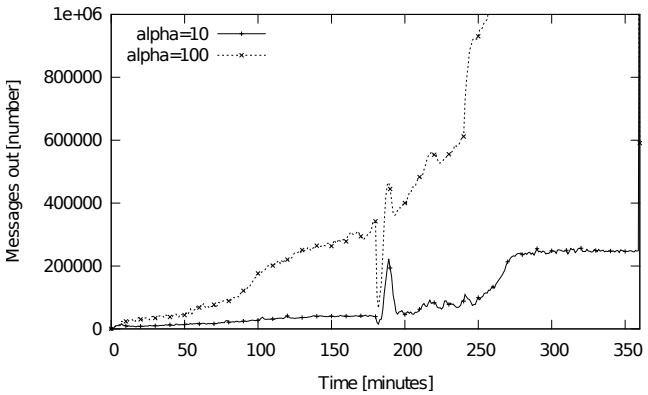
(a) E.1: 8 Instances, Messages.



(b) E.1: 16 Instances, Messages.

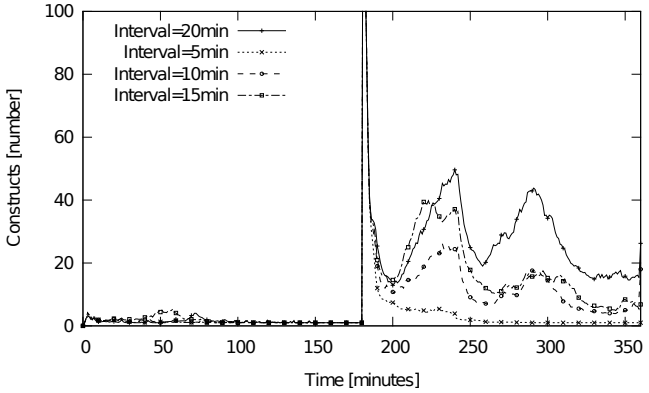


(c) E.1: 32 Instances, Messages.

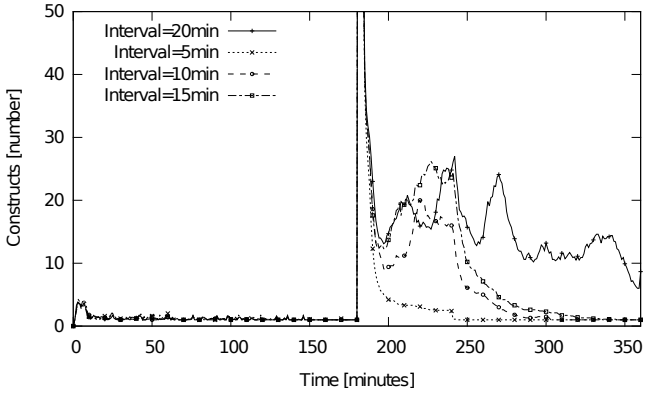


(d) E.1: 64 Instances, Messages.

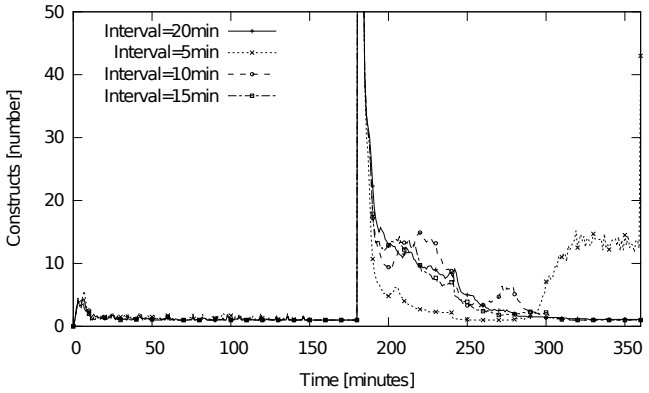
Fig. 19. E.1: Ring Reunion Algorithm with parallel instances, $\alpha \in \{10, 100\}$, message consumption.



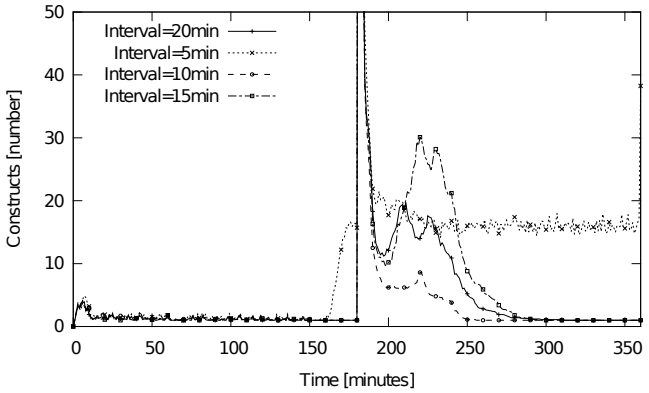
(a) E.2: 8 Instances, Constructs.



(b) E.2: 16 Instances, Constructs.

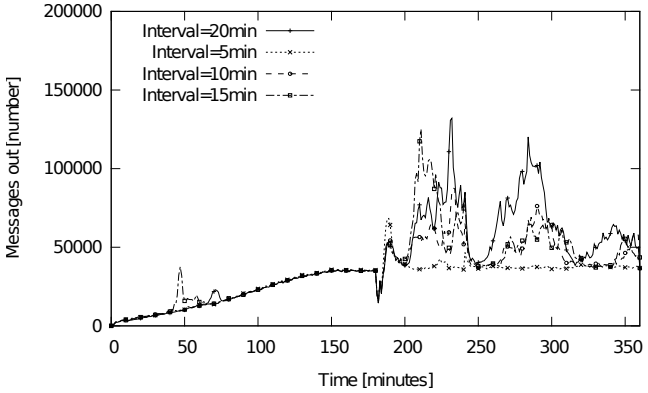


(c) E.2: 32 Instances, Constructs.

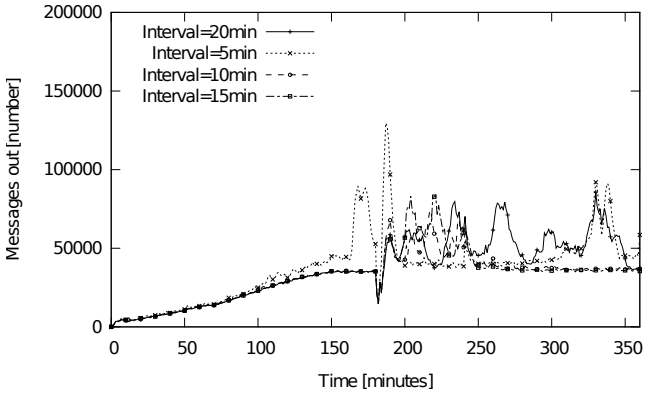


(d) E.2: 64 Instances, Constructs.

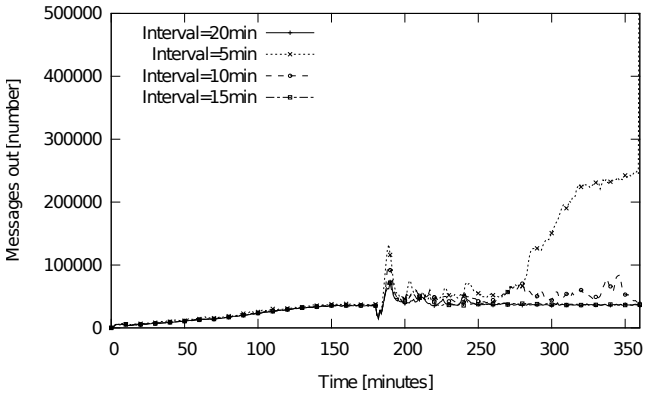
Fig. 20. E.2: Ring Reunion Algorithm with parallel instances and various values for interval parameter, $\alpha = 10$, constructs during merger.



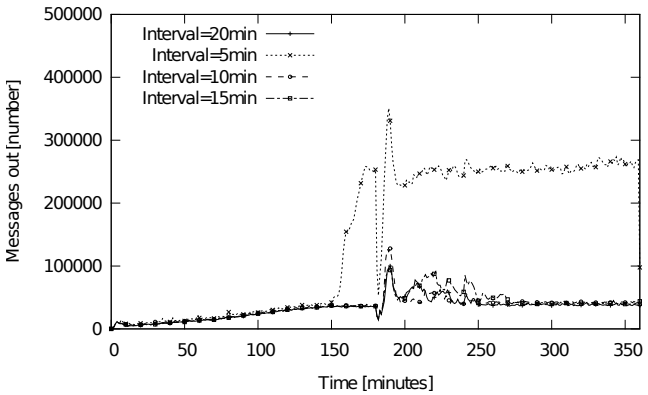
(a) E.2: 8 Instances, Messages.



(b) E.2: 16 Instances, Messages.



(c) E.2: 32 Instances, Messages.



(d) E.2: 64 Instances, Messages.

Fig. 21. E.2: Ring Reunion Algorithm with parallel instances and various values for interval parameter, $\alpha = 10$, message consumption.