



An Empirical Study of the Global Behavior of Structured Overlay Networks as Complex Systems

RUMA R. PAUL

Licentiate Thesis in Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

TRITA-ICT 2015:12

ISBN 978-91-7595-700-5

Abstract

Distributed applications built on top of *Structured Overlay Networks (SONs)* operate based on certain self-* behaviors of the underlying Peer-to-Peer network. Among those, self-organization and self-healing are the two most prominent and assumed properties. The operating environment of distributed systems continues to be more inhospitable with the advance and demand of new technologies; for example in case of mobile and ad hoc networks *Churn* (node turnover) can be extremely high due to node mobility, frequent disconnects/reconnects and configuration changes. Also, in such dynamic environments, the system may face high *Churn (node turnover)* and *Network partition* in a frequent manner. The situation becomes worse if the self-healing behavior of underlying SON is not complete and well defined. This implies the following non-trivial questions: Can the maintenance mechanism of a SON heal the damage to the structure due to harshness of the operating environment and reverse it back? What are the pre-conditions; in other words, what properties the healing mechanism should possess in order to achieve reversibility against stressful environments? Existing literature lacks such assessment and verification study of the self-healing property of a SON.

In this thesis, we investigate both the behavior and design of a system that operate in inhospitable environments. This work is relevant to systems with both peaks of high stress (e.g. partitions, churn, network dynamicity etc.) and continuous high stress. We evaluate existing overlay maintenance strategies, namely Correction-on-Change, Correction-on-Use, Periodic Stabilization, and Ring Merge. We define the *reversibility property* of a system as its ability to repair itself to its original state. We propose a new strategy, called *Knowledge Base*, to improve conditions for reversibility against inhospitable environments. By means of simulations, we demonstrate reversibility for overlay networks with high levels of partition and churn. We make general conclusions about the ability of the maintenance strategies to achieve reversibility.

Identification of *Phase Transitions* in a SON can provide useful information about the properties of each state of the system. Also, this enables to find the critical points in the operating space and parameters influencing them. The applications running on top of the SON can potentially utilize this knowledge to adapt its operation accordingly in different system states. In this thesis, a representative ring-based SON, namely *Beernet* is chosen and extended to achieve reversibility. The resulting overlay, *Beernet++* exhibits reversible phase transitions under churn. We analyze the critical points observed during such transitions. We present the behavior of *Beernet++* for high level of churn and network partitioning, along with their interaction.

Acknowledgments

I am deeply thankful to the following people, without the help and support of whom, I would not have managed to complete this thesis:

My supervisors Peter Van Roy and Vladimir Vlassov, for their guidance, constant feedback and encouragement throughout this work.

My former colleague at UCL, Sébastien Doeraene, for his assistance with the Mozart programming system, in which all the experimentation are conducted in this work.

Boris Mejías Candia, for the discussion and assistance with the system, Beernet, which is outcome of his Ph.D. thesis, used for the experiments in this work.

Jérémie Melchior, who is using the enhanced Beernet, which we have termed as Beernet++, for implementing his system, a Distributed Mobile UI, for his continuous assistance towards making Beernet++ well defined.

My former colleague, Nicholas Rutherford for reviewing initial work and good discussion.

Vasiliki Kalavri, for sharing her experiences with me, for reviewing and giving useful tips regarding presentation of the papers.

The anonymous reviewers of the papers, for providing observant and constructive feedback.

My EMJD-DC colleagues and colleagues at UCL and KTH, especially Manuel Bravo, Zhongmiao Li, Navaneeth Rameshan, Amin Khan, Hooman Peiro Sajjad, Paris Carbone, Vamis Xhagjika, Jingna Zeng, Leila Sharifi, Solomon Liu and Petar Mrazovic for always being available to discuss ideas, for being supportive and for the good times we spent hanging out together.

My family, for their continuous support, their faith in me, their patience and immeasurable understanding.

Stockholm, 20 October 2015

Ruma R. Paul

Contents

List of Figures

1	Introduction	1
1.1	Peer-to-Peer Systems	2
1.2	Self-Management, Self-Stabilization and Reversibility	3
1.3	Problem Definition	4
1.4	Thesis Contribution	5
1.5	Thesis Organization	6
2	Background	7
2.1	Representative Complex Systems	7
2.2	Beernet	8
3	Operating Space; Maintenance and Behavior	13
3.1	Global Operating Space	13
3.1.1	Churn	14
3.1.2	Network Partition	14
3.1.3	Network Dynamicity	15
3.1.4	Workload	15
3.1.5	Ring Size	15
3.2	Overlay Maintenance Strategies	15
3.2.1	Correction-On-Change and Correction-On-Use	15
3.2.2	Periodic Stabilization	17
3.2.3	ReCircle	17
3.2.4	Knowledge Base	18
3.3	Behavior of a SON	20
4	Failure Detection and Replica Management	23
4.1	Failure Detector	23
4.1.1	QoS Metrics for Eventually Perfect Failure Detection	25
4.1.2	Evaluation	25
4.1.3	Related Work	28
4.2	Transactional DHT	29
4.2.1	Lazy Data Migration	30

4.2.2	Evaluation	31
4.2.3	Related Work	31
5	Investigation about Churn	33
5.1	Reversibility	33
5.2	Evaluation of Reversibility	34
5.2.1	Correction-On-Change and Correction-On-Use	35
5.2.2	ReCircle (Periodic Stabilization and Merger with Passive List) . . .	36
5.2.3	Knowledge Base for Each Node	37
5.2.4	Discussion	38
5.3	Evaluation of High-Level Properties	38
5.3.1	Damage and Recovery of Ring Topology	39
5.3.2	Data Level Parameters	39
5.4	Related Work	40
6	Phase Transitions	41
6.1	Definition of Phase, Phase Transition and Critical Point	42
6.2	Observation of Phase Transitions	43
6.2.1	Increasing Churn with Time	43
6.2.2	Continuous Moderate Churn	45
6.2.3	Gradual Increase and Decrease of Churn	46
6.3	Related Work	47
6.4	Discussion	48
7	Investigation about Network Partitioning	49
7.1	Types of Partition	49
7.2	Evaluation of Maintenance Strategies	50
7.2.1	Execution During Network Partition	51
7.2.2	Execution at Partition Repair (Network Merge)	53
7.3	Related Work	58
7.4	Discussion	59
8	Interaction between Network Partitioning and Churn	61
8.1	Stranger Model	61
8.2	Evaluation of Maintenance Principles	64
8.2.1	Correction-on-*	64
8.2.2	Correction-on-* and Periodic Stabilization	64
8.2.3	ReCircle (Periodic Stabilization and Merger with passive list) . . .	65
8.2.4	Knowledge Base	65
8.2.5	Oracle	65
8.3	Recovery Time and Cost	66
8.4	Related Work	67
8.5	Discussion	67

9	Conclusions and Future Work	69
9.1	Reversible and Predictable System	69
9.2	Future Work	70
9.2.1	API and Phase Transitions	70
9.2.2	Maintenance Principles and Efficient Self-Healing	71
9.2.3	Network Dynamicity and its Impact	71
9.2.4	Experimentation and Validation on Real-World Environment	71
	Bibliography	73

List of Figures

1.1	Overlay Network: A P2P System with nodes a, f, i, p and x forms the overlay network on top of the underlay network	3
2.1	General Structure of Beernet	9
2.2	Branches on a relaxed ring. Peers p and s consider u as successor, but u only considers s as predecessor. Peer q has not established a connection with its predecessor p yet.	10
2.3	Three steps of Join Algorithm: contact the successor, contact the predecessor and acknowledge the join.	11
4.1	Detection and Reaction Time for various values of k	27
4.2	Accuracy for k	27
4.3	Accuracy for m	27
4.4	Detection and Reaction Time for various values of m	28
4.5	Data Level Parameter: % of failed transactions	32
4.6	Data Level Parameter: % of lost updates	32
4.7	Data Level Parameter: % of lost keys	32
4.8	Data Level Parameter: % of inconsistent replicas	32
5.1	% of nodes on core ring as a function of time (in sec) after withdrawing churn to assess reversibility. Figure 5.1a, 5.1b and 5.1c are not reversible (nodes on core ring never converges to 100%). Figure 5.1d using Knowledge Base is reversible.	34
5.2	% of lookups and joins which remain incomplete after injection of churn for 1 minute	36
5.3	% of incomplete joins with time during injection of churn for 1 minute	36
5.4	Properties for increasing churn after injecting a particular churn value for 1 minute	39
5.5	% of failed transactions	40
5.6	% of lost keys	40
6.1	Phase Transitions in Beernet++: red, green and blue (dark, gray and light-gray in B/W) areas correspond to % of nodes on ring, branches and isolation respectively	44

6.2	Phase Transitions in Beernet++ under low churn (0% to 5%): red, green and blue (dark, gray and light-gray in B/W) areas correspond to % of nodes on ring, branches and isolation respectively	45
6.3	Phase Transitions in Beernet++: red, green and blue (dark, gray and light-gray in B/W) areas correspond to % of nodes on ring, branches and isolation respectively	46
6.4	Phase Transitions in Beernet++ due to increasing and decreasing churn: red, green and blue (dark, gray and light-gray in B/W) areas correspond to % of nodes on ring, branches and isolation respectively	47
7.1	Two different types of partition scenarios: white and black nodes belong to two different partitions	50
7.2	Two partition scenarios: white and black nodes belong to two different partitions; partition having black nodes have absence of more than $ succ_list - 1$ consecutive peers (here, $ succ_list = 4$).	52
7.3	Number of islands as a function of time (in sec) starting at the moment of sparse partition repair to assess self-healing using different maintenance strategies	55
7.4	Number of Messages generated for 2, 4 and 10 sparse partitions using different maintenance mechanisms	56
7.5	Number of islands as a function of time (in sec) starting at the moment of sequential partition repair to assess self-healing using different maintenance strategies	57
7.6	Number of Messages generated for 2, 4 and 10 sequential partitions using different maintenance mechanisms	58
8.1	Evaluation of Stranger model for 10%, 30% and 80% of churn	63
8.2	Number of islands as a function of time (in sec) after withdrawing churn and partition to assess self-healing against strangers using different maintenance strategies	64
8.3	Number of islands as a function of time (in sec) after withdrawing churn and partition to assess self-healing against strangers using different maintenance strategies	66
8.4	Recovery/Healing time for increasing strangers	67
8.5	Number of messages generated for increasing strangers	67

Chapter 1

Introduction

The advent of Internet has given rise to software systems in which a set of autonomous computers connected through a network, communicate and coordinate their actions as well as share resources. Such *Distributed Systems* are perceived as a single, integrated computing facility by the users. This suggests *distribution transparency*, however, network and computer failures breaks transparency and complicates programming of such systems. Thus, handling partial-failure through complete recovery is a key challenge in building reliable distributed systems. Also, as the number of nodes increases, achieving scalability is another issue. The initial approach of building distributed systems is the simple client-server architecture. Though this is still a popular way, the server(s) becomes a single point of failure and congestion, thus violates both fault-tolerance and scalability of the system. The improvement of network bandwidth and computing power has led to the way of edge-computing. The *Peer-to-Peer (P2P)* systems are one such approach, which make use of resources available at the edge of a network, thus has become a popular way of conceiving distributed systems. There are also social factors behind such popularity, as people are willing to connect more to other people and share resources in order to achieve mutually beneficial exchange. Thus, distributed systems are becoming larger and complex.

A large-scale distributed system consists of many interacting parts and their overall behavior cannot be predicted in a straightforward way from the behavior of each part. They have many operating modes depending on the environment in which they work and what they are supposed to do. These characteristics of distributed systems inspire approaches to consider them from the point-of-view of *Complex Systems*. In such systems, complex structures or behaviors at macro-level emerge from simple interactions or behaviors of the various subsystems at the micro-level [1], [2], [3]. In a P2P system, it is necessary to identify the macro-level behaviors emerge due to the interactions and actions of individual peers. This chapter provides a brief discussion on P2P systems and describes the problem addressed in this thesis.

1.1 Peer-to-Peer Systems

The basic idea of P2P systems is the dual client/server role of each node/peer of the system, which allows taking advantage of the resource available at the edge of the network. The widely known first P2P system is *Napster* [4], a file sharing service, which allows users to exchange files directly instead of via a server. *AudioGalaxy* [5] and *OpenNap* [6] are two other systems, which belong to first generation of P2P networks. However, all these systems are not entirely peer-to-peer, they are based on mixed architecture and still require a server to work. Peers connect to a server with a query about files. In reply, the server provides the addresses of the peers storing the requested file. The peer then directly connects to the responsible peers to download the file. In case of server failure, it is not possible to make new queries, but the exchange of file can continue. The centralization of search operation and administration allow easier shut-down of such system, as Napster ceased its operation in 2001 due to legal issues.

The second generation of P2P systems is the first completely decentralized ones. Gnutella [7] and Freenet [8] are the main two representatives of this generation. These systems do not rely on any server to work. Each peer establishes and maintains some local connections; search queries are forwarded using these connections and replied by the responsible peer(s). Due to such local cooperation of participating nodes an overall network routing view emerges, which is known as an *overlay* network, on top of the underlay network. Usually the underlay network is the Internet, which is used by the overlay network for routing of messages. Figure 1.1 shows an example overlay network formed by a P2P system with 5 participants.

For the second generation of peer-to-peer networks, the overlay formed is unstructured, i.e. peers connect to each other in a random manner, without any pre-defined topology. As a result, this approach suffers from high query overhead, as search queries are flooded through the network. This raises concerns about scalability of the systems. Also, if flooding is done with time-to-live, there is no guarantee of successful termination of a query. All these problems of unstructured overlay networks have made the way for a structured solution, i.e. *Structured Overlay Networks (SONs)*, the third generation of P2P systems. A structure is induced through the pointers maintained by each peer of the system, i.e. the structure of the emerged overlay is a macro-level property based on the micro-level states of the participating peers. An identifier space is embedded into the graph of a structured overlay, where each node is assigned a unique identifier from this space and also responsible for certain identifiers. Adding structure to Peer-to-Peer networks has provided efficient routing, guaranteed reachability and consistent retrieval of information; however also introduced the challenge of maintaining the structure. Among all the structures proposed for SONs, the ring topology is the most popular choice. As mentioned in [9], ring topology is competitive with other *SONs* in terms of reaching any other node in smaller steps and also most resilient to failures. Unfortunately, along with these good properties, the temporary inconsistency of the ring structure poses several challenges for correct operations. Many *SONs* were proposed to gradually improve and circumvent or relax the requirements of perfect ring for accuracy; *Chord* [10],[11], *DKS* [12], *Beernet* [13], [14] to name a few.

1.2. SELF-MANAGEMENT, SELF-STABILIZATION AND REVERSIBILITY

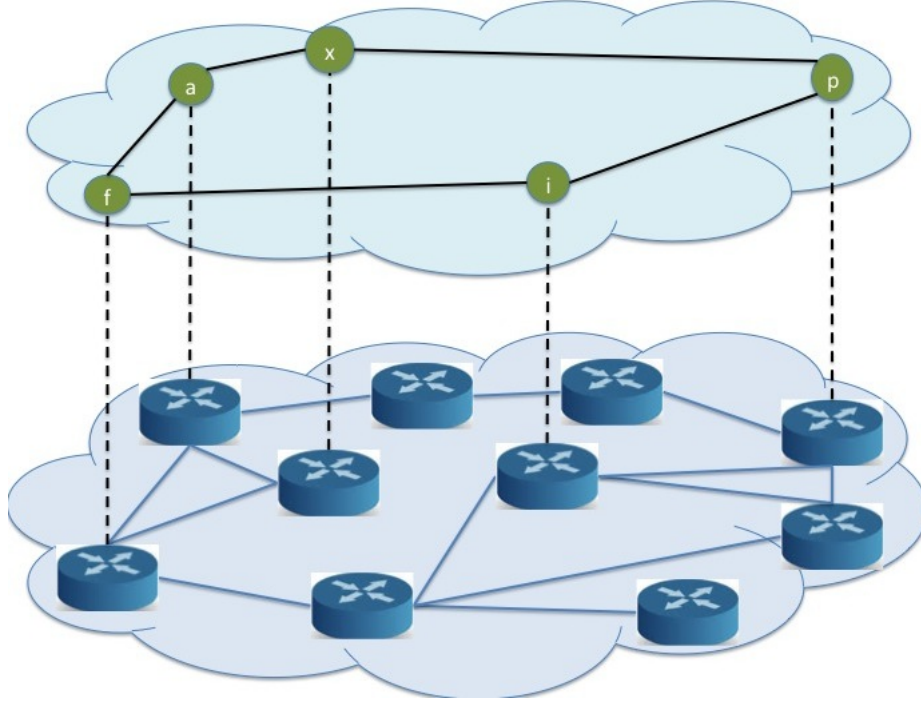


Figure 1.1: Overlay Network: A P2P System with nodes a , f , i , p and x forms the overlay network on top of the underlay network

1.2 Self-Management, Self-Stabilization and Reversibility

The unreliability and heterogeneity of edge machines, along with different network technologies introduce a crucial challenge for P2P systems, as the size of the system grows, they become more and more difficult to manage. The conventional management becomes very difficult, time-consuming, and error-prone. In order deal with high-level complexity, enhancing *Self-Management* capacities of the system has no alternative. The term “Self-Management” implies the ability of a system to modify itself, as per the high-level management policies, to handle changes in its initial state or environment, without any human intervention. The objectives of self-management are typically classified into four categories: self-configuration, self-healing, self-optimization and self-protection [15]. Together they are referred in literature as self-* properties.

Recently much work has been conducted on the concept of *Self-Stabilization* [16], [17] as a non-masking fault-tolerance for distributed systems. A system is self-stabilizing iff: i) irrespective of the initial state, it will eventually converge to a correct state, ii) once it reaches a correct state, it is guaranteed to stay in a correct state, provided no fault happens. These two properties enables a self-stabilizing distributed algorithm to recover from any transient failure. Here, a transient failure implies violation of a system’s state, but not the behavior of the system [16].

The *Structured overlay networks (SONs)* have become a popular way of implement-

ing large scaled distributed systems. The most prominent reason is that the fully decentralized architecture of SONs can easily be made self-organizing and self-healing. Such self-management properties are crucial to deal with the inherent complexities introduced by decentralization. Also, the operating environment of distributed systems continues to be more inhospitable with the advance and demand of new technologies. Assurance of reversibility through complete self-healing is the only way to survive in such increasingly stressful environments. We define the *Reversibility* property of a system as its ability to repair itself to its original state when the external stress is withdrawn. In other terms, the functionality of a system is a property of current environment hostility and not of the history of environment hostility.

Our concept of reversibility is related to, but different from the concept of self stabilization. A self-stable system is able to survive arbitrarily high levels of transient failures, i.e., self-stabilization assumes perturbation of node state, which is not the case discussed in this thesis. Reversibility implies that overall functionality, at the system level, depends only on the inhospitability of the current operating environment and not on the history of stresses the system had experienced. In other terms, past catastrophes do not break the system. There is also another subtle difference. Reversibility in a SON might also concerns about permanent failures, where nodes fail and new nodes join at a rate that is defined as *Churn*. Thus, the system is dynamic, i.e., churn is causing the system to change over time. When looking at the overall system, high churn will cause certain functionalities to disappear and when churn decreases, they come back. Therefore there is an analogy between high churn and temporary failures, at the system level, even though at the individual node level there are only permanent failures.

1.3 Problem Definition

Distributed applications break down when there are too many node failures or communication failures. Typically, such applications revert to an “offline mode” with reduced functionality in this case. This is sometimes acceptable for applications that have a client/server architecture, such as mobile applications that depend on a data center. The data center remains a single point of failure. However, this is now changing as the Internet is becoming more and more decentralized: data centers are increasing in number and come in many different sizes. Applications running on such an infrastructure need to have a decentralized architecture that is resilient to failure. Ideally, the application should survive with partial functionality during arbitrary system failures and recover its full functionality when the underlying system is restored. This is not just a fringe case: mobile and ad-hoc networks, for example, have this kind of failure. Even supposedly stable parts of the Internet have peaks of unstable behavior.

Due to the inherent complexity introduced as a result of de-centralization, large-scale distributed systems, in particular P2P systems have brought forward many non-trivial challenges. These systems face many problems, especially when they are stressed beyond where their behavior is a straightforward extrapolation of the behavior of their parts, i.e. the emergence of macro-level behavior. The experience of all who have attempted to build

1.4. THESIS CONTRIBUTION

such large-scale distributed applications is that they are very difficult to get right: they require continual babysitting by teams of specialists to keep them running. The reason behind this is that, like all highly available systems, these systems are designed to operate within a specific failure model and threat model and their behavior is typically undefined outside these models. The standard approach based on redundant fault tolerant algorithms has reached its limit.

This problem can only be solved by a new approach. We propose an approach to build applications that are able to survive arbitrary failures, providing reduced but predictable functionality in that case, and when the failures go away the application fully recovers its functionality. In other terms, to design systems such that a change in operating conditions can cause a qualitative change in system behavior according to a well-defined transition instead of performing unpredictably. Our goal is to design the system to work in a well-defined manner for the complete phase space, i.e., for all possible operating conditions including extremely inhospitable ones. We define an “inhospitable environment” as one in which certain stress parameters (such as churn, node failure rate, communication delay) can potentially reach high values and temporarily increase without bound. The goal is to build systems that are both predictable (hence, useful in practice) and reversible (hence, they survive) in these environments. This is important for three essential reasons. First, for practical system design it is important to explore highly stressful environments, since even systems running in so-called “stable” environments will have peaks of high stress. Second, it can open new venues for application design, such as mobile and ad hoc networks, for which current fault-tolerance techniques are insufficient. Third, it is important for scientific reasons, to understand what happens in highly inhospitable regimes.

1.4 Thesis Contribution

The objectives of this thesis is study of both the behavior and design of complex systems, in particular a class of *Structured Overlay Networks (SONs)* defined by the reference architecture of [18], in inhospitable environments. As a representative of this class we have chosen Beernet for our practical experiments. Given its generic design, the results obtained for Beernet should be qualitatively the same for other members of the class. The overall contributions are as follows:

- As prerequisites of the study, organization of the entire operating space of the representative complex systems using a set of stress parameters. Definition of “behavior” of the system using a set of matrices. As part of making the system well defined in entire operating space: i) Integration of a QoS-aware self-adaptable eventually perfect failure detection algorithm; ii) Augmentation of replica management with an optimistic lazy-data migration protocol for better availability and consistency of replica sets.
- Definition of “reversibility” concept: the functionality of a system is a property of current environment hostility and not of the history of environment hostility.

- Introduction of Knowledge Base as a new maintenance mechanism next to existing mechanisms: Correction-on-change, Correction-on-use, Periodic stabilization and Over- lay Merge;
- First demonstration of reversible SON and identification of conditions needed to make a SON reversible, under a wide range of churn values;
- Reversible phase transitions: experimental demonstration in a SON with churn (A phase is a qualitative description of the structure of the SON. A phase transition occurs when a significant fraction of a system's parts changes phase.);
- Investigation of network partitioning. Identification of preconditions for partition-tolerance and reversibility for any scenario and level of network partitioning;
- Investigations of interaction between churn and network partitioning: Identification of the limits of the combination of partitioning and churn, up to which the system is able to achieve reversibility by itself, using a model of how the partitions diverge with time (the Stranger Model).
- Demonstration of a reversible SON and identification of conditions needed to make a SON reversible while facing network partition for any duration, with or without churn in between;
- Comparative analysis of self-healing of different maintenance strategies in inhospitable environments.

1.5 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 describes our representative class of complex systems using reference architecture for overlays. In Chapter 3, we discuss existing and our contribution in maintenance strategies of SON, along with the organization of operating space of a SON. This chapter also presents the set of metrics to define the behavior of a SON. The enhancements introduced to our representative system to make it well defined are discussed in Chapter 4. Chapter 5 and Chapter 7 present our investigation about churn and network partitioning respectively. In Chapter 8, we explore the interaction between the two stress parameters, namely network partitioning and churn. Finally, we summarize our contributions and discuss future works in Chapter 9.

Chapter 2

Background

For practical reasons, we have chosen a particular class of complex systems to conduct our study in this thesis. We build on the concept of *Structured Overlay Network (SON)*, a well known approach to building decentralized distributed systems. As per the reference architecture, proposed in [18], a class of SONs is our representative complex system. For the practical experiments, we have chosen Beernet [13], an overlay with properties typical of this space. In this chapter, we describe key aspects of our chosen class of SONs and Beernet.

2.1 Representative Complex Systems

According to the reference model proposed in [18], there are six key design aspects, using which any overlay network can be characterized. Here, we provide a brief description of this design space, along with the specification of our representative class of overlays (which we will refer as ring overlays). The reference architecture is general enough to include many overlays: Chord [11], Chord# [19], SkipNet [20], DKS [12], Koorde [21], P2PS [22], Beernet [13], Mercury [23], [24], EpiChord [25], Accordion [26], Symphony [27]. Our contributions can be generalized for other overlays having similar key aspects in the design space.

- **Choice of Identifier Space:** virtual identifier space I , having some closeness metric $d : I \times I \rightarrow \mathbb{R}$. For ring overlays identifier space is a subset of \mathbb{N} , of size N , with $d(x, y) = (y - x) \bmod N$.
- **Mapping to the Identifier Space:** $F_P : P \rightarrow I$ associates peers with a unique virtual identifier from I and $F_R : R \rightarrow I$ associates resources with identifiers from I . For ring overlays, F_P can be a uniform hash function or some random function, also can be order preserving, as in DKS. A virtual identifier is assigned to a peer when it joins the overlay and this mapping remains static. F_R is similar to F_P for ring overlays, usually it is a uniform hash function, which distributes resources uniformly in the identifier space, thus provides implicit load balancing.

- **Management of the Identifier Space:** $M : I \rightarrow 2^P$ associates with identifier of a resource r , $i = F_R(r) \in I$, the set of peers managing r . Each peer p is responsible for the set $M^{-1}(p)$ of identifiers. For ring overlays, a peer with virtual identifier p is responsible for the interval $(predecessor(p), p]$. In these approaches, the responsibility of a peer may dynamically change due to churn in the overlay.
- **Graph Embedding:** a directed graph, $G = (P, \varepsilon)$, where P is the set of peers and ε denotes the set of edges. A neighborhood relationship $N : P \rightarrow 2^P$, for a peer p , $N(p)$ is the set of peers with which p maintains a connection. The graph formed by ring overlays complies with Kleinberg's small-world principles [28], thus belongs to the special class of "routing-efficient" small-world networks. In these overlays, each peer p perceives the identifier space to be partitioned into $\log(N)$ partitions, where each partition is k ($k = 2$ for Chord) times bigger than the previous one. The routing table of p contains $\log_k(N)$ connections to some nodes from each partition.
- **Routing Strategy:** a non-deterministic function $R : P \times I \rightarrow 2^P$, which at peer p , with neighborhood $N(p)$, for a target identifier i selects the (set of) next peers $R(p, i) \in N(p)$ to forward the message. Due to small-world networks, decentralized, greedy routing strategy provides the best performance in ring overlays. So for a target identifier i , peer p selects the closest preceding link, $d \in N(p)$ to forward the message. Since, there are always k intervals, routing converges in $O(\log_k(N))$ hops.
- **Maintenance Strategy:** a maintenance strategy is required to maintain the structural integrity while peers go offline or network connection fails. Joining is handled explicitly by all overlays using a join protocol, whereas leaving/failure is implicit, thus requires a maintenance strategy to maintain the connectivity of the underlying graph. As Aberer et al. [18] point out: "The practical usability of an overlay network critically depends on the efficiency of the maintenance strategy." The goal of this work is the enhancement of maintenance in ring overlays, so that these overlays can survive inhospitable environments and achieve reversibility.

2.2 Beernet

In our study, we use the Beernet [13] system for experimentation. This is a straightforward SON that supports all the maintenance principles. It is an example of the reference architecture (see Section 2.1). Also, it is a non-trivial complex system with interesting global behavior, and a practical scalable transactional store. Beernet is designed so that ring maintenance operations are extremely simple. The join operation in Beernet is done in two steps: isolated node \rightarrow node on branch \rightarrow node on core ring. Each step is a simple node-node communication; no locking is needed. Communication failures can cause nodes to be ejected or ringlets to form. Figure 2.1 shows a general structure of Beernet. The neighborhood of each peer in Beernet consists of successor, predecessor, connections/fingers in routing table (as described in Section 2.1), a successor list and a predecessor list.

2.2. BEERNET

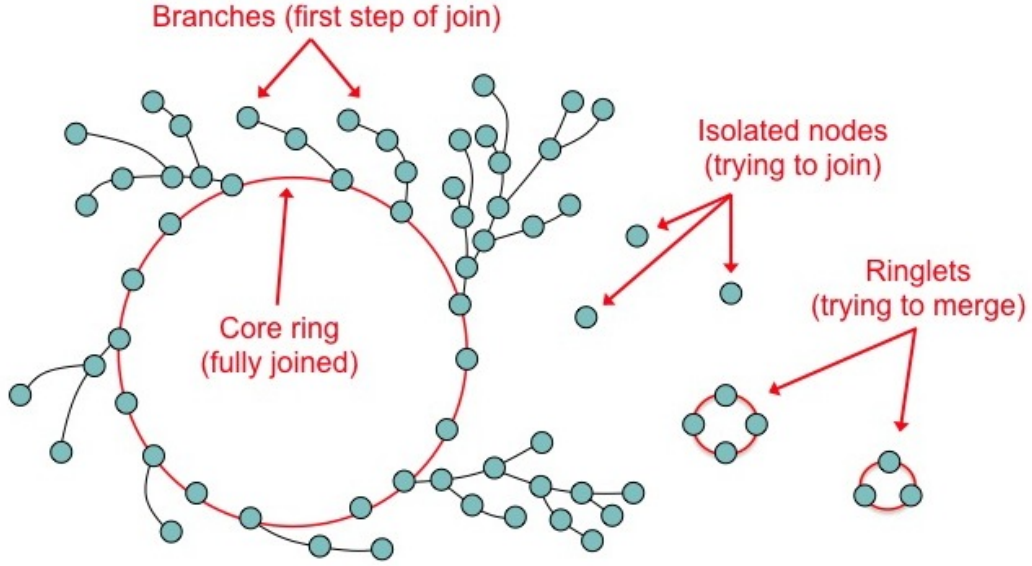


Figure 2.1: General Structure of Beernet

The successor list looks ahead along the ring and contains maximum $\log_k(N)$ peers. The predecessor list contains all the peers that consider the current node as their successor.

Ring with Branches / Relaxed Ring: Beernet has two invariants: *Every peer is in the same ring as its successor* and *A peer does not need to have connection with its predecessor, but it must know its predecessor's key*. The first invariant allows a new peer to be part of the network by connecting only to its successor. The second one determines the responsibility of a peer. These two properties allow relaxation of the ring: when a peer is not still connected to its predecessor it forms a branch from the core ring. Figure 2.2 shows a Beernet network, where red nodes are organized into a ring and green nodes are on branches. The branch rooted at peer u is created, because peer q still has not made any connection with its predecessor, p ; so peer p is not aware about q yet. In the meantime another peer s has joined between q and u , so the size of the branch has increased.

The routing principle of Beernet is: *a peer p always forwards the lookup request to the responsible candidate*. This is a slight variation of Chord's routing mechanism to support relaxed-ring architecture. This routing strategy makes sure that p doesn't miss any peer in between its successor and itself. Due to introduction of branches, the guarantees about proximity offered by Beernet lookup mechanism corresponds to $O(\log_k(n) + b)$, where b is the distance to the farthest peer on the branch.

Join needs no locking: To join a relaxed ring, a new peer triggers a lookup request using its own key to find the best successor candidate. After the current responsible peer of the key replies, the new peer triggers the join process. The atomic join algorithm of relaxed ring consists of 3 steps, each step involving only 2 peers. The join process is shown in

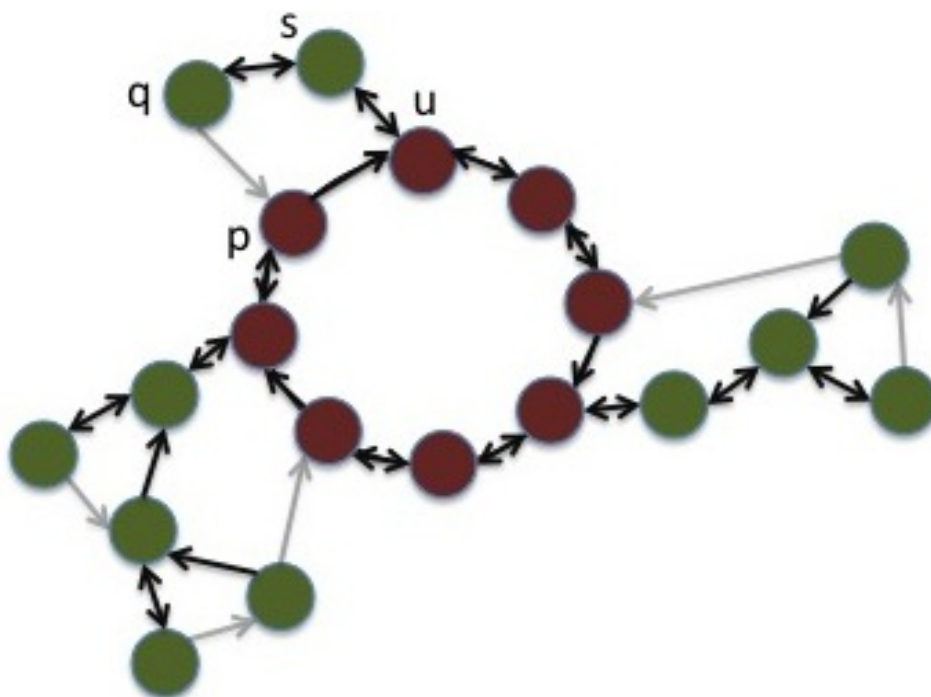


Figure 2.2: Branches on a relaxed ring. Peers p and s consider u as successor, but u only considers s as predecessor. Peer q has not established a connection with its predecessor p yet.

Figure 2.3, where a node q joins the ring in between peers p and r , $q \in]p, r]$. It is noted that, after the 1st step, peer q is on a branch, as it has not yet established a connection with its predecessor and q will continue working on a branch (where r is the root of the branch) if it fails to establish a connection with p . During the last 2 steps, there are no changes of responsibility, so the address space is already consistent after the first step.

2.2. BEERNET

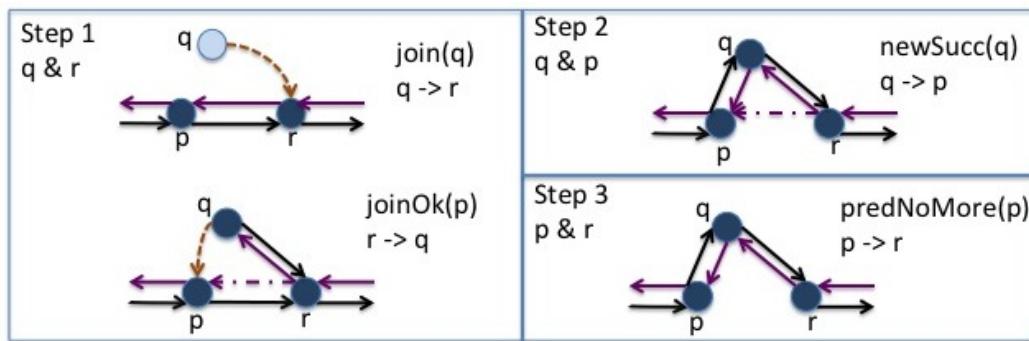


Figure 2.3: Three steps of Join Algorithm: contact the successor, contact the predecessor and acknowledge the join.

Chapter 3

Operating Space; Maintenance and Behavior

The objective of this thesis is a study of complex systems in its entire operating space, in particular the inhospitable environments. As a prerequisite, it is necessary to identify the stress parameters, which create such inhospitality. In order to achieve fault-tolerance against inhospitable environment, maintenance strategy is an integral part of each *Structured Overlay Network (SON)*. This chapter presents our proposed organization of the operating space of a SON. Next, we describe the maintenance strategies of ring-based SONs, both existing and our contribution. These principles can be classified along two dimensions: local/global and reactive/proactive. Compared with previous work, we add one new principle, called *Knowledge Base*. With this new principle, our set of principles covers all points in the two-dimensional space. Finally, this chapter concludes by identifying a set of high-level properties to define the behavior of a ring-based SON.

3.1 Global Operating Space

The operating space of a *Structured Overlay Network (SON)* is large. In order to conduct an exhaustive study it is essential to organize it. Most of this space is constituted by the scenarios when something goes wrong, so we can also call this organization as the fault model for a SON. The main focus is to identify all possible non-malicious failure scenarios that a SON may face during its lifetime: given n nodes connected by communication links, any set of nodes may crash, any communication link or set of links may slow down or fail, permutation of these failure scenarios can lead to infinitesimal possibilities. In this work, we have attempted to structure all such possible failure conditions using 5 dimensions. Any point in this 5-dimensional space represents a valid operating environment that a SON may face during its lifetime. The rest of this section will define each of these dimensions and describe the parameters that characterize them.

3.1.1 Churn

Churn is the most usual and basic scenario that a SON faces during its lifetime. The term churn is used to express the measurement of peers joining or leaving the network during a given period of time. The literature of SON mostly addresses churn as the only failure scenario in order to establish the basic fault tolerance and self-management characteristics. However, in most cases churn remains under a certain limit. Studies such as [29, 30] show that a peer-to-peer environment experiences high churn rates, where nodes continuously join and leave the system and their up-time in the overlay is short. Several studies [31], [32] have been conducted to understand peer behavior or churn in different peer-to-peer systems. As per [31], the majority of peers are long-lived, however the remaining short-lived peers join and leave the system at high rate, which comprises a large portion of sessions. According to this, we can say that even systems with low/average churn face high peaks. So, it will be interesting to observe the behavior of a SON under excessive churn.

We present the definition of churn, used in this work. As nodes join and leave during churn, with equal probability for join and leave events, we can say that a node changes its identity during the churn with only two events. In other terms, if we assume equal probability of join/leave event and a single event per time unit, then every 2 time unit, a node will leave and a new node will join the network, i.e., every 2 time unit the total number of peers on the network will be same, whereas only a single node has a changed identity. We assume that churn varies over time, and that the average number of correct nodes at any instant is constant. We have defined churn as the percentage of nodes turnover per time unit.

3.1.2 Network Partition

Network partition is one of the worst scenarios that a SON might face during its lifetime. During the partition of underlying network, the nodes of a SON are divided into multiple disjoint sets, where a node can communicate with the nodes of its own set, but is unable to contact the nodes in the other sets. Any long-running large distributed system is bound to come across network partitions during its execution. Several reasons may cause the underlying network to partition: link failure, router issues (failure, misconfiguration, overloading), malicious activities (denial of service attacks), software bugs or physically damaged network equipment [33],[34], [35]. For example, as a consequence of several natural disasters, it was exposed that the global network connectivity depends on a few active "choke points" [36]. Failure of such choke points under natural calamity or similar events can lead to network partitions [33]. Apart from these, policy based issues or conflicts can also cause inaccessibility across regions, resulting in network partitions [37],[38]. Though partition can be seen as massive churn, but it creates distinguishable operating condition for a SON by itself, that's why deserve to be a separate dimension in our proposed global operating space. The only parameter, which characterizes this dimension, is the number of partitions, i.e., the number of rings in the system for ring-based SON.

3.2. OVERLAY MAINTENANCE STRATEGIES

3.1.3 Network Dynamicity

Network dynamicity happens when communication links slow down, resulting in congestion in the network. The impact of congestion leads to false suspicions in a SON. This poses challenges, as each false suspicion event triggers failure recovery, routing table update and other mechanisms. Huge numbers of frequent false suspicions create a challenging operating condition for a SON, in the face of which, the SON struggles to maintain its structure. We define congestion in our fault model, as the percentage of node (or link) connectivity change per time-unit. By the term “node connectivity change” we mean the increased or decreased time experienced by the rest to make a contact with that particular node. The link connectivity change signifies the increased or decreased time experienced by the nodes on the both side of the link, to contact the other.

3.1.4 Workload

The application workload often proves to be a key factor in determining the hostility of the operating environment. A particular amount of workload may work quite well for a particular SON, whereas might make another limited resource SON unresponsive. The workload can be defined as the number of transactions per time-unit along this dimension.

3.1.5 Ring Size

The last dimension of the proposed operating space of a structured overlay network is the size of the network, the number of nodes in the system. This dimension in fact evaluates the SON’s scalability.

3.2 Overlay Maintenance Strategies

Following [39], we classify maintenance strategies of overlays into proactive periodic correction (e.g., Chord, Chord#) and reactive mechanisms, e.g., correction-on-change and correction-on-use (as used in DKS, P2PS, Beernet). Also, an overlay merge algorithm has become necessary part of a SON. In our work we have adapted a partition-merging algorithm, *ReCircle* [40]. *ReCircle* has two parts: a periodic stabilization algorithm and a reactive *Merger* that gets triggered in case of extreme events. Along with these known strategies, we have introduced *Knowledge Base*, which is essential to survive inhospitable environments. Table 3.1 summarizes the properties of these principles. We describe each principle and its integration with Beernet. We refer to the resulting extended system as *Beernet++*.

3.2.1 Correction-On-Change and Correction-On-Use

These principles were introduced by DKS (we will refer to these principles together as *Correction-on-**). Correction-on-change is concerned about join/leave/failure of nodes. Whenever such events are detected, successor and predecessor lists are updated and the correction of pointers (successor, predecessor and fingers) is triggered. Correction-on-use

Principles	Local/ Global	Reactive/ Proactive	Fast/ Slow	Safety	Bandwidth Consumption
Correction-on-*	Local	Reactive	Fast	Yes	Small
Periodic Stabilization	Local	Proactive	Slow	Lookup inconsistencies and uncorrected false suspicions can be introduced	High
Merger with Passive List	Global	Reactive	Adaptable	Yes	Adaptable
Merger with Knowledge Base	Global	Proactive	Adaptable	Yes	Adaptable

Table 3.1: Ring Overlay Maintenance Principles

mainly corrects the fingers. Every time messages are routed, information is piggybacked to correct fingers. As a result more network usage makes the routing table more accurate. Thus, correction-on-use provides self-optimization and self-configuration, whereas partial self-healing is achieved through correction-on-change. These two principles are fast in terms of reacting and updating local state for events like join/leave/failure, which is important to survive an inhospitable environment without introducing any inconsistency. However, without any event, no healing or maintenance is done (i.e., lack of liveness), making these mechanisms restricted form of self-healing.

Beernet [13], our representative SON, uses both mechanisms for overlay maintenance. As healing is completely dependent on detection of join/fail events, performance of the failure detector plays a crucial role. A QoS-aware eventually perfect failure detection is integrated [41], as described in Chapter 4. When a node suspects another peer, it updates its own successor and predecessor list. If the suspected peer is the successor of the current node (p) then it adjusts its successor pointer by choosing the first one in the successor list (suppose r) and triggers the recovery mechanism by sending a *fix* message to r . When r receives a *fix* message from p , it makes p the new predecessor if its current predecessor is suspected or p is a better predecessor than the current one. The last step of the failure recovery is the *fixOk* message, which is triggered by r , after *fix* is accepted. In this step, p fixes and propagates the successor list. In case of a false suspicion, the failure detector triggers an *alive* message. Now, there are 3 cases to consider; the falsely suspected peer is: i) a better predecessor, in that case the predecessor pointer is corrected and the peer is added to the predecessor list; ii) a better successor, which now requires the correction of the successor pointer and addition to the successor list, also a *fix* message is triggered to run the protocol, this is needed in the case when the old successor also falsely suspects the corresponding peer; iii) any other peer, in that case no special action is required. In all these cases the falsely suspected peer is removed from the crashed list.

3.2. OVERLAY MAINTENANCE STRATEGIES

3.2.2 Periodic Stabilization

Chord uses this simple idea of periodic correction mechanism for ring maintenance, where each peer periodically checks the validity of its predecessor/successor pointers. Periodically a peer asks its successor about the predecessor of the current successor. If it is the same as itself, it does nothing; however if it is a new node, then it is probable that this new node is a better successor for itself. Thus by exchanging periodic messages with its successor a node attempts to maintain its immediate vicinity. However, this proactive mechanism might become a slow response while facing an inhospitable environment. As discussed in [42], lookup inconsistencies and uncorrected false suspicions can be created in real implementations. Also, [43] analyzes that inconsistencies can appear in Chord (which uses periodic stabilization for ring maintenance) because of churn. According to [44], for a ratio of churn to stabilization frequency, while doing a lookup the longest finger of any peer is always found to be dead, which degrades routing efficiency. In order to avoid this, it is required to trigger periodic stabilization often, making an inefficient use of bandwidth. Thus, we can say that this proactive mechanism is complementary to reactive correction-on-* principles: the correction-on-* require no extra messages in case of no failures, but do not work when no event is detected, whereas periodic stabilization needs no event to execute (i.e., it has liveness), but is costly in terms of bandwidth consumption.

Beernet++ incorporates this periodic maintenance mechanism. As described above, every δ time units each node p checks for a better successor candidate. To do this, p asks its current successor q about its predecessor pointer and updates if it finds a clockwise closer node to p than q . On the other hand, q also tries to update its predecessor pointer if p is a better candidate than the current one.

3.2.3 ReCircle

ReCircle [40] extends periodic stabilization to react to extreme events like network partitions and merge. It has two parts: periodic stabilization, as described before, and merger. Periodic messages are issued to maintain the local geometry, whereas the merger issues messages that navigate further, triggering awareness and remedying the anomalies, thus ensuring eventual convergence towards one ring. Once the overlay converges, the messages issued by the merger die out and ReCircle behaves as a normal periodic maintenance algorithm. Here we provide a brief sketch of the merger, as adapted in Beernet++.

Each node maintains a queue, which holds the identifiers of all nodes that need to be fixed, i.e., areas that violate the ring's geometry and introduce inconsistencies. The queue may become non-empty due to churn, network partitions, network congestion, flash crowd and so forth. Every γ time units each node m dequeues the elements from its queue. For each element n , m generates an event called $mlookup(n)$, to fix a possible inconsistent location n on the identifier space. Furthermore, along with $mlookup(n)$ by m , n also makes an $mlookup(m)$. Each $mlookup(id)$ performs a greedy routing to the problem area defined by id , which is similar to a normal lookup operation. Once it reaches the peer responsible for id , it tries to fix the ring by triggering the same mechanism as periodic stabilization. Further $mlookups$ are generated to carry on the repairing of the ring in the clockwise direction.

Each *mlookup* spreads this fixing process by generating new *mlookups* for random identifiers on the ring. This is done by enqueueing *id* into the queue of random nodes selected from the routing table of the current node. This is controlled by a knob, called the *fanout* parameter, to trade-off bandwidth consumption and convergence time. Thus merger can be made adaptable in term of convergence time. Also, an *mlookup* tries to fix any wrong successor/predecessor pointers while routing. Shafaat et. al. in [40] propose “passive list” mechanism to populate the queue at each node. At each node, a list of nodes, called passive list, is maintained that consists of all nodes currently being suspected by that node. Whenever a node is no longer suspected, it is enqueued into the node’s queue. Thus, this is a reactive approach to trigger the merger.

3.2.4 Knowledge Base

In this thesis, we introduce *Knowledge Base (KB)* as a new maintenance mechanism next to existing ones, already described. The idea of KB serves two purposes: to provide necessary knowledge for the completion of joining of new peers and to trigger the merger of ReCircle in a proactive manner. As the churn intensity increases the frequency of unsuccessful join attempts increases. The first step of the join protocol of a SON is to do a lookup for successor and after receiving a response, the new peer becomes part of the SON. With the increased churn, lookup failure rate also increases, resulting in pending joins where new peers keep on waiting to receive responses of their join requests (see Section 5.2). This creates isolation for the new peers. There are two ways to make such isolated nodes be part of the SON: i) by adding such nodes to the node queues on the overlay, thus triggering the merger of ReCircle in a proactive manner; ii) by providing a valid join reference and re-triggering the join request. As isolated nodes are still not part of the SON, the nodes on the overlay have no reference to these nodes, thus no healing mechanism will be effective. In order to apply any of these proposed solutions the knowledge about the alive peers on the overlay is needed. Also, triggering the merger only in a reactive manner is not sufficient to provide complete self-healing in an inhospitable environment. As the churn intensity increases, the overlay might be partitioned in such way that there is no communication problems among the partitions. This might also happen as well due to network partitioning (discussed in detail in Chapter 7). For such scenarios, no suspicion event gets issued. As a result, the passive list mechanism used in the ring merge algorithm [40] fails to trigger the merging.

When there is simultaneous network partition and churn, the number of strangers (refer to Chapter 7) increases due to the interaction between these two stress parameters, merger with passive list fails to achieve reversibility (refer to Chapter 7). In order to ensure reversibility up to the best limit, for such scenarios and initiate merging of overlays in the same partition, require more knowledge than a passive list to trigger the merger, which calls for building of a *Knowledge Base (KB)* at each peer. The knowledge base at each node is the best-effort view (complete/partial) of the global membership of the system. It extends ReCircle [40] in two ways: i) consider bigger set of nodes than the passive list, ii) proactive triggering of merger: even if there is no false suspicion (explained shortly). The nodes in the KB are not monitored by the failure detector at a peer, thus causes no change

3.2. OVERLAY MAINTENANCE STRATEGIES

of the embedded small-world graph of the overlay. The knowledge base at each node can be accessed through an API. There can be different levels of knowledge base:

- **Passive KB:** Such KB can be built through listening only. At peer p , for each node $a \notin KB_p$ that p comes across (while routing or as a member of its current neighborhood), the virtual identifier and the network reference of a is added to KB_p . Building of knowledge base at each node is based on passive observation; the list built by this strategy is never shared with other peers, thus creates no impact on bandwidth consumption or scalability of the overlay. However, such KB may be out of date quicker.
- **Active KB:** Using this strategy a node communicates with others to enhance its KB. This can be a weak algorithm, such as each joining node just informing others of its existence, or a stronger algorithm, such as gossip where each node asks a random node to send its KB that it unions with its own KB. The result is that the KB converges “faster” to maximum information, so that when a partition arrives, it is in the best possible condition for surviving (least possible strangers). The effect of the active approach is mainly on the convergence time of KB. However this approach causes extra load on available bandwidth.
- **Oracle:** This is information coming from “outside the system”. In order to use the first of two alternative solutions proposed above to deal with isolated nodes, an oracle is required. Also, as the duration of the underlying network partition increases, more and more nodes in each partition are replaced by new nodes due to churn. Thus, eventually the nodes in one partition will become complete strangers for the nodes on other partition, as $(\bigcup_{p_i \in P_1} KB_{p_i}) \cap P_2 = \emptyset$ (and vice versa), here P_1 and P_2 are the sets of peers of two partitions. This may also happen for short/arbitrary duration of partition depending on the intensity of churn. In Chapter 8, we discuss about the stranger syndrome using an analytical model. For such scenarios, knowledge base built at each node falls short to merge the overlays as the partition ceases. As two overlays have no knowledge about each other, the system requires the intervention of third-party to gain reversibility. An oracle can be part of the API, using which the application layer or a bootstrap server can give information to the system.

In order to use the KB efficiently, it is necessary to define a proactive sampling technique that periodically chooses elements of the KB. Since the KB can be very large, it is not practical to use all elements of the KB. Every σ time unit, each node randomly picks up an element e from its KB, where e is neither member of its current neighborhood nor is currently suspected, and enqueues e into its queue. Here, σ is the mean of a Poisson distribution of times when a periodic sampling happens and is a tunable parameter, which affects the convergence time of the overlay merging. This mechanism can be made more efficient by making σ adapt to the operating environment, which we leave as future work.

The KB at each node is always growing. It may or may not converge to cover all nodes in some partitions, depending on the operating conditions. But the number of “dead” node references in the KB will always grow, which introduces garbage collection problem. We can add a time-stamp to all references in KB and optimistically remove all “sufficiently

old" nodes from the KB, implementation and validation of this is left as future work. The knowledge base presented in our work is a generalization of the passive list proposed in [40], which only keeps track of currently suspected nodes. As a node only monitors the members of its current neighborhood, so can only trace failures in this set. So, we can say that the knowledge base is a super-set of the passive lists at any time, thus enriches the nodes with required knowledge to trigger overlay merge even for long-term partition.

Knowledge Base and Gossip Frameworks

In a general gossip framework, each peer maintains a state (local knowledge of the overall system), and by some rule (e.g., periodically, on demand, etc), each peer communicates with other peers by exchanging the state, and updating it based on the new knowledge that was discovered in the received state from another peer (e.g., if the state is about global membership of the system, then each peer can accumulate a certain sample of the network depending on the size of the state). Furthermore, the way peers decide to contact one another also could be either simply random, or driven by the locally accumulated knowledge (e.g., in T-Man [45] peers try to contact not uniform random peers, but those that are known and have closest ID).

The *Knowledge Base* principle presented in Section 3.2.4 closely resembles a gossip framework. However as already mentioned, the knowledge gathering at each peer can be passive or active. In order to construct an active KB, a peer can use a gossip algorithm, whereas for passive KB, a peer accumulate knowledge based on observation and also from past and current neighborhood of the peer. In this thesis, for the experiments we have used passive KB. Whether integrating an active gossip protocol attain any improvement is subject to our future work. Also, we intend to improve the maintenance and application of knowledge at each node by borrowing the ideas from gossip protocols.

3.3 Behavior of a SON

We define the term “behavior”, with respect to a SON using a set of metrics, which capture the behavior of a particular SON at various levels. For a systematic and thorough understanding of the behavior of a SON with transactional DHT, we have identified 3 levels. Below we present the parameters of all levels, studying which will provide an assessment of the impact of a particular environment on the operation of a SON:

- Data Level: % of failed transactions, % of lost keys, % of inconsistent replicas, % of lost updates.
- Connection Level: Number of times imperfections introduced, % of time a node experiences imperfections, % of nodes on core ring.
- Routing Level: Number of messages generated.

The data level parameters signify the impact of a particular environment on the DHT and data level operations. The last parameter, % of lost updates provides useful information

3.3. BEHAVIOR OF A SON

for application running on top of a SON regarding the ratio of successful updates of the stored data in a particular environment. This parameter is mostly dominated by % of failed transaction, however other scenarios may also contribute. Suppose, the replica set of key k_i is formed by peers a, b, c, d , where a, c, d has the latest value with version 7 (i.e. there were total 7 updates) for k_i and b holds the old value with version 5. Now, if a, c, d leave or crash or become unavailable (for example, due to partition), then the old value stored in b will be replicated to the new responsible for key k_i . In this scenario though there was no failed transaction, but 2 updates are lost. Another scenario: suppose transaction is designed for an application in such way that the update operation of a key depends on its old value, in this case if the key is lost, then it doesn't invoke any update, contributing to the lost updates. All other data level parameters are apparent from their titles.

In all ring-based SONs, each peer keeps a successor and a predecessor pointer to maintain the structure. The connection level parameters assess the deviation of the ideal ring structure during adjustment with a particular environment. The parameter *Number of Imperfections Introduced*, counts the total number of times peers falsely suspect their ideal successor or predecessor pointer and choose an imperfect peer. The next parameter, *% of time a node experiences imperfections* accounts for how long a peer goes through such imperfection. These 2 parameters reflect the impact of false suspicions on the ring. The parameter *% of nodes on core ring*, tries to measure the rigidity of the ring. In other terms, it finds out the maximal ring in the system and reports % of nodes on it. The routing level parameter *Number of messages generated* shows how many messages are generated to cope up with the changed environment. As already mentioned, these metrics are general enough to be applied for any SON with transactional DHT.

Chapter 4

Failure Detection and Replica Management

As prerequisites of our study, we have enhanced some aspects of our representative *Structured Overlay Network (SON)*, Beernet. As part this, we present a QoS-aware self-adaptable failure detection algorithm [41]. Also, we have augmented replica management of Beernet with an optimistic, low cost data migration protocol as part of failure recovery algorithm, in order to achieve improved availability and consistency of replica sets [41].

4.1 Failure Detector

Being a distributed system, partial failures are usual incidents for Beernet, which it must cope with. As the maintenance strategy of Beernet is Correction-on-Change, *Failure Detectors* play an important role in Beernet's operation, because most of these changes (nodes crashing or leaving the network) are triggered by the failure detection module at each peer. Perfect failure detection is impractical especially for Internet style networks. An *Eventually Perfect Failure Detector* is the best that can be achieved on the Internet. Along with Strong Completeness, an eventually perfect failure detector requires *Eventual Strong Accuracy*, which implies that any false suspicion of failure will be eventually corrected. This is feasible to implement because, if the unreachability, thus failure suspicion, was caused by problems of the underlying communication link (failure or slow), it will eventually go away as the link recovers, thus be able to communicate with the suspected peer. This is the kind of failure detection that Beernet's algorithms rely upon, however, *Quality of Service (QoS)* of failure detection modules is crucial for Beernet's performance, as frequent false suspicions may cause instability in the SON. Also longer period to detect a crash or correct a false suspicion will allow greater inconsistency in the system. For these reasons, maintaining QoS of failure detection is of utmost importance. This section presents an implementation of self-adaptable eventually perfect failure detector, which adapts with the environment, providing QoS.

The existing public release of Beernet (version 0.9) [46] includes a generic implementation of an eventually perfect failure detector, guided by that in [47]. The idea is very

simple: each peer p periodically sends a *ping* message to all other peers in Π , where Π is the set of peers p knows about. After the ping message is sent, p launches a timer, which corresponds to the timeout for responses from all peers in Π . Each node q , which receives a ping message from p , immediately responds with a *pong* message. When p receives a pong message from q , q is added to *alive* set. When the timeout is triggered, all the peers, which are not in the alive set are suspected to have failed. Then p starts a new round and repeats the same way of collecting responses from other peers. If a pong message is received from a peer r , where $r \in \text{suspected}$, implies that r is falsely suspected in earlier round. This is detected at the next timeout, when r belongs to both *alive* and *suspected* sets. In this case the false suspicion regarding r is corrected and the timeout is increased by a fixed value, Δ in order to prevent such false suspicions.

The algorithm above has two problems as we observed. The first issue is, the timeout is global for all connections of a node whereas the round trip time is very often different for each connection. For this global timeout, the detection process of a peer is driven by the speed of the slowest connection. Let us consider the function of a failure detector of a particular peer a , where a monitors b and c . The connection between a and b is very fast but between a and c is very slow. In the first round, a will receive response from b within due time but will falsely suspect c . Later, when a will receive response from c , it will increase its timeout, this way a will continue increasing its timeout period until there is no false suspicion, meaning that the timeout is adapted to the slowest connection. The second problem with the above algorithm is that it only increases the timeout to adapt to the round trip time (RTT) between two nodes, whereas the RTT may vary wildly along time, in a long running distributed system. Due to temporary congestion, RTT may go higher, leading to false suspicion or increase of timeout and after some time when the congestion goes away it may drop to the normal previous value. Thus, it is necessary to lower the timeout period to adapt to the RTT, without creating any oscillation.

The expectations from the failure detector module are threefold: to reduce the number of false suspicions, to correct mistakes quickly, and also to detect failures quicker. Unfortunately, these goals are contradictory: the time to detect failures or correct a mistake depends on the timeout period. Lower timeout delays lead to quicker detection of failures or correction of mistakes, but in order to reduce the number of false suspicions the timeout delay needs to be sufficiently large enough to cope with slight changes in the environment. In order to determine the perfect or best value of timeout period for each connection adapting to the RTT of each connection is the only option.

Based on above principle, we present self-adapting failure detection, which adapts with RTT of a connection and also keeps a safety period to accommodate the variability in network connection. Each node maintains a history of last k RTT for each connection and the timeout period of next round for each connection is determined based on this history. We will explain this using an example of a pair of nodes (as for each connection the procedure is same), where a is monitoring b (i.e. connection $a < - > b$). At each round, a sends *ping* message to b , each *ping* message is time-stamped. When b receives a *ping*, it replies with a *pong*, including the timestamp of the corresponding *ping*. As soon as a *pong* message from b is received, a calculates the RTT of connection $a < - > b$ for this round. After the timeout, if no *pong* message from b is received, a starts suspecting b , if b is not already in

4.1. FAILURE DETECTOR

suspected set. However, if b is in *suspected* set and a *pong* is received from b in current round, then the false suspicion regarding b is corrected. After deciding the status of b for the current round, a starts preparing for the next round of connection $a < - > b$ by calculating the timeout period.

Timeout Period for next round = Average RTT over last k rounds + m *Standard Deviation of RTT over last k rounds.

The first part of the above equation is used to estimate the expected value of RTT of the next round. However, due to unpredictability of a large and complex network like Internet, a safety period is needed so that in case of sudden changes of environment, the timeout period can cope up quickly. For this purpose, a weighted standard deviation over the history of RTT is the best candidate, where the weight m depends on the variability in the network. Together, as a summation of both quantities, enables to determine the most suitable value of timeout period for the next round and also auto-adapts with the changing environment. This failure detection algorithm can be expressed as a function of k and m , the best values for these parameters can be determined for a particular environment, based on experiments, as shown in Section 4.1.2. For the initial rounds, when there is not enough RTT data collected, a pre-defined timeout period is used. The algorithm of the self-adapting failure detection module is shown in Algorithm 1.

4.1.1 QoS Metrics for Eventually Perfect Failure Detection

As already mentioned, QoS of the failure detection module plays an important role in Beernet's performance or resilience against a particular environment. The expectations from failure detection module are threefold: quick detection of failures, reduce the number of false suspicions or mistakes and correct a mistake as soon as possible. From a direct mapping of these expectations, we define 3 primary QoS metrics for an eventually perfect failure detector, which are in-line with those proposed by Chen et al. [48]:

- **Detection Time:** Time to detect a failure or crash. This is the average duration after which all peers permanently suspects the crashed node.
- **Accuracy:** % of suspicions which are correct i.e. % of suspicions when the suspected node has actually crashed/left the system.
- **Reaction Time/Mistake Duration:** Time required to correct a false suspicion. This is the average duration after which a peer becomes aware of a false suspicion and corrects its mistake.

4.1.2 Evaluation

The objective of this section are threefold: i) To evaluate the failure detection algorithm using the QoS matrices as described in Section 4.1.1, ii) To understand the impact of the two parameters (k and m) on the QoS of the failure detection in a given environment and ii) To tune these two parameters for upcoming experiments. The ideal value of these parameters

Algorithm 1 Self-adapting Failure Detector

```

upon event  $\langle \text{monitor} \mid p \rangle$  do
  MonitoredPeers := MonitoredPeers  $\cup \{p\}$ 
  p.waitPeriod := INIT_DELAY
  send  $\langle \text{ping} \mid \text{self}, \text{timestamp}_{\text{ping}} \rangle$  to p
  starttimer(p.waitPeriod)
end

upon event  $\langle \text{timeout} \mid p \rangle$  do
  if  $p \in \text{alive}$  and  $p \in \text{suspected}$  then
    suspected := suspected  $\setminus p$ 
    trigger  $\langle \text{alive} \mid p \rangle$ 
  if  $p \notin \text{alive}$  and  $p \notin \text{suspected}$  then
    suspected := suspected  $\cup p$ 
    trigger  $\langle \text{suspect} \mid p \rangle$ 
  alive := alive  $\setminus p$ 
  expectedRTT := Average of last k values in p.RTTHistory
  stdRTT := Standard Deviation over last k values in p.RTTHistory
  p.waitPeriod := expectedRTT + m*stdRTT
  send  $\langle \text{ping} \mid \text{self}, \text{timestamp}_{\text{ping}} \rangle$  to p
  starttimer(p.waitPeriod)
end

upon event  $\langle \text{ping} \mid q, \text{timestamp}_{\text{ping}} \rangle$  do
  send  $\langle \text{pong} \mid \text{self}, \text{timestamp}_{\text{ping}} \rangle$  to q
end

upon event  $\langle \text{pong} \mid p, \text{timestamp}_{\text{ping}} \rangle$  do
  RTT = CurrentTime - timestampping
  p.RTTHistory := p.RTTHistory  $\cup$  RTT
  alive := alive  $\cup p$ 
end

```

4.1. FAILURE DETECTOR

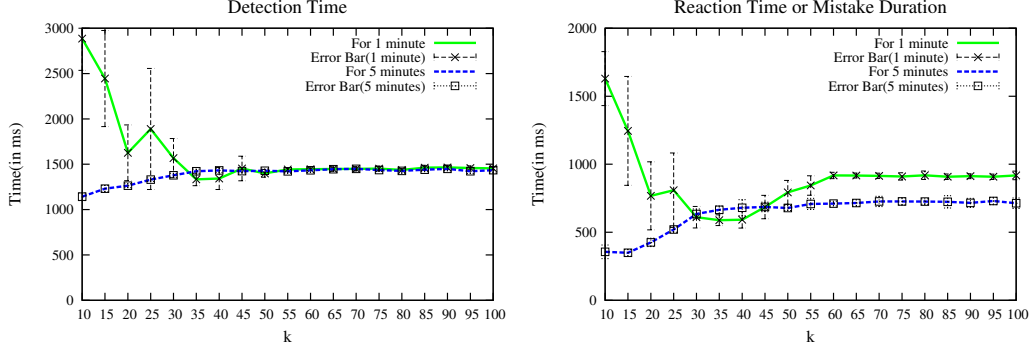


Figure 4.1: Detection and Reaction Time for various values of k

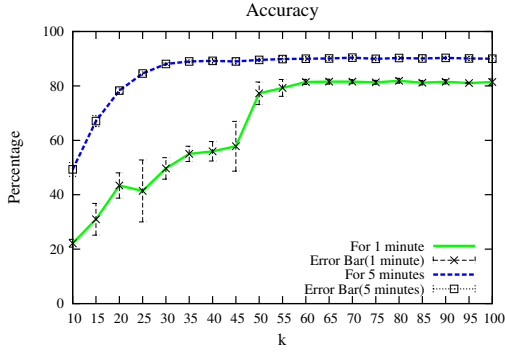


Figure 4.2: Accuracy for k

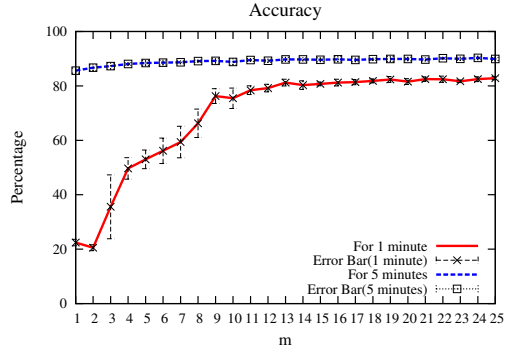
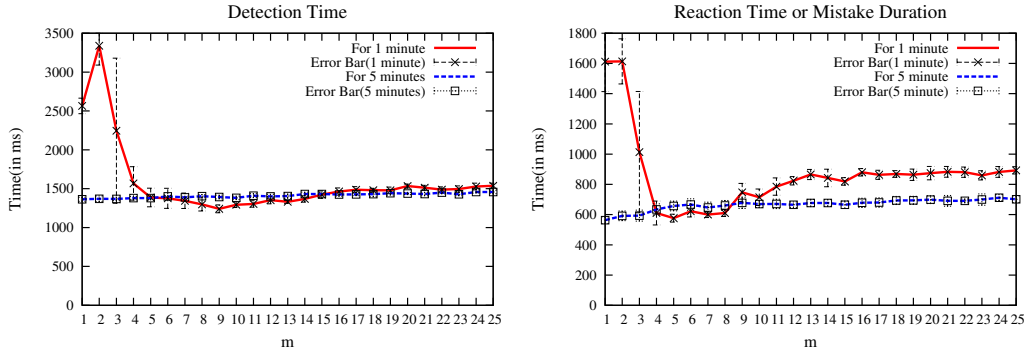


Figure 4.3: Accuracy for m

to achieve best performance is highly dependent on the environment. It is possible to make the failure detector as self-tuning, but we leave this as future work.

To evaluate the QoS of failure detection, a network of 100 peers is used. Correction-on-* principles are used as the sole maintenance strategy. To simulate the underlying network, the end-to-end delays are set based on the empirical distribution of minimum RTT provided in [49]. During the steady state of the SON, 5% churn is injected. The churn events are modeled as a *Homogeneous Poisson Process (HPP)* with λ_1 events/sec, where $\lambda_1 = 10$ for 5% churn on a network of 100 peers. Simultaneously, in order to simulate variability in the underlying network, 5% links (in this work link always refer to end-to-end connection of the overlay) changes connectivity every 5s, based on the empirical distribution of standard deviation in per-connection RTTs as measured in [49]. The connectivity change events are also modeled as a HPP with arrival rate of λ_2 events/sec, where $\lambda_2 = L/100$, here L is the total number of connections in the system. Also, to evaluate the adapting capability of the failure detection scheme, we have taken measurement for 2 different duration, 1 minute and 5 minutes.

For this given environment, we have measured the QoS parameters for various values of k , by keeping m fixed as 4, the result obtained is shown in Figure 4.2 and Figure 4.1. The accuracy increases with buffer size until it reaches a maximum point and remains same

Figure 4.4: Detection and Reaction Time for various values of m

both for short and long durations. As the environment is changed consistently throughout the experiment, longer history provides more statistical information, however in case of temporary fluctuation of environment, storing very long history might create negative impact on QoS, also buffer size becomes an issue. As it is apparent, after adapting to the environment for longer time, accuracy is improved for the same buffer size. In terms of detection and reaction time, we get good results for buffer size of 30-40, beyond that we again get an increasing trend for short duration. However for longer experiments, the outcome increases till buffer size of 30 – 35 as accuracy goes up, after that it remains steady as it already have a good estimate of the RTT and settles for the detection and reaction time for the best accuracy. For the following experiments, buffer size is set as 30, as this is sufficient history size for the failure detector to reach the optimal state.

The same environment is used to test the impact of m on QoS of failure detection; Figure 4.3 and Figure 4.4 portray the result. As expected, accuracy increases with increased timeout period, i.e., with increment of m for short experiment. However, for longer period, failure detector's adapting capability is prominent, as the average RTT dominates the timeout period, i.e., accurately measures the expected RTT which leads to maximal accuracy, thus, making the impact of m fades away. This happens because network variability is modeled as a HPP, however for an unpredictable environment like Internet, m will have its impact even for longer up time. The detection and reaction time decreases sharply for $m > 3$, remains almost same for 4 till 9 and again shows an increasing trend for short experiment duration. Based on this analysis, we have kept $m = 4$ for the remaining experiments in this work in order to cope up with the environment right from the beginning.

4.1.3 Related Work

There are 2 implementation strategies for timeout based failure detectors [50, 51]: i) Heart-beat based, where each process periodically sends “I am alive” messages to all other processes; ii) Ping based, a process monitors another process by sending a ping message and asking for acknowledgement. The ping strategy provides finer-grained control, also according to design philosophy of Beernet, each node has a set of other peers, which it monitors, and this matches the pull strategy. However, existing work on QoS of a failure detector

4.2. TRANSACTIONAL DHT

is mostly based on heartbeat-based implementation, the reason as mentioned in [50], is the quality of the estimation of the timeout period. In contrast to the heartbeat strategy, in ping strategy the number of variables is twice (for example, transmission, reaction delays). To maintain QoS, these approaches [50, 48] use sampled arrival times to compute an estimation of the arrival time of the next heartbeat. The timeout is set according to this estimation plus a safety margin (which is constant in [48], whereas [50] computes it by using Jacobson's algorithm[52]). Very few works are found on the ping based failure detectors. A message efficient algorithm is provided in [53], but in this approach each process monitors only a single process. In [54], the failure detection is lazy, where the status of the monitored process can be known by making a query to the failure detection module. A process is suspected when the elapsed time for the unacknowledged message is more than the maximum round trip time till observed. However, the maximum round trip time can be much higher than the average response time due to a spike, so this may increase the detection time. The protocol provided in [55] assumes a model of finite average response time; timeout increases logarithmically with the total number of slow messages (triggering false suspicions) and linearly with the number of fast messages (acknowledgements received before timeout) since the last slow message. Whenever, a slow message arrives, the number of fast messages is set to zero and therefore results in a drop of the timeout (if the number of fast messages was greater than zero). This may cause increased false suspicions as the timeout decreases after a false suspicion. Also this is a pull mechanism to get the status of a monitored process, whereas Beernet needs failure detector to push notifications regarding any event.

4.2 Transactional DHT

Beernet provides support for Transactional Distributed Hash Table (DHT) with replication. The performance or correctness of the applications running on top of Beernet using the Transactional DHT is dependent on the assurance provided by the data layer operations of Beernet. This section provides a brief description of data layer support in Beernet, along with an augmentation of replica management in order to achieve improved availability and consistency among the replicas of a particular data item.

There are three protocols implemented in the transactional layer of Beernet: Two-phase commit, Paxos Consensus Algorithm and Eager Paxos, in order to cover the requirements of various types of application. Though Two-Phase commit is the most popular approach used by traditional databases, it has two stringent requirements: survival of the transaction leader and all replicas must be updated when the transaction completes. These two requirements are hard to be satisfied in a dynamic environment. However, Paxos Consensus protocol relaxes these requirements by using replicated transaction managers and quorum-based commits, thus provides fault tolerance without sacrificing strong consistency. So, in this work, only Paxos Consensus protocol is used in order to evaluate the data layer parameters, as the obtained result can also be applied for Eager Paxos.

Symmetric Replication [43] is implemented in Beernet as a replication strategy. The replica management layer of Beernet strives to maintain a consistent set of replicas under

any extreme environment. Let us analyze the operations of this layer in detail under churn. When a new peer joins, it replaces its successor as member of the replica sets of a certain amount of items. During the join procedure, the successor pushes all such data-items to the new peer that it has become responsible for. Taking values only from a single replica is fine in this case, as if the successor has stale value of a data-item, this will replace a bad replica with another bad replica. However, when there is a failure, it is more important to read from the majority during the recovery, as there is no way for the recovery node to know whether the dead peer was up-to-date or not. In the existing public release of Beernet [46] the replica management layer only handles the join events and [13] analyzes the replica management under churn, concluding that failure recovery requires reading from the majority. In order to read from majority, doing transaction for each data-item becomes too expensive and false suspicions make the situation more complicated. In this work, we have augmented the replica management layer of Beernet, by proposing an inexpensive lazy-migration protocol during failure recovery, which can achieve eventually consistent replica set for a data-item, without creating any conflict with simultaneous transaction of that particular data-item.

4.2.1 Lazy Data Migration

Each data-item has a version number associated with it, which only increases by each update. The *read* operation during transaction reads from the majority and returns the value with the highest version number. While committing a write operation, a replica votes for commit if $version_{new} \geq version_{existing}$ for an existing data-item or the data-item is absent. The monotonicity of version number, along with the advantages of symmetric replication is exploited in lazy migration protocol. Suppose, peer q has successor r and predecessor p , when p suspects q , it initiates failure recovery, now if q is also suspected by r , r takes over the responsibility for all the data items that q was responsible. Due to symmetric replication strategy, r can find out other members of the replica set of the corresponding data items using the symmetric function, so there is no need to add any expensive group management to the replica sets. Though there is a replica set per data-item, due to symmetric replication, many replica sets overlaps, which facilitates the data migration. Lets consider the replica set is formed by a , e and m (for replication factor 4 and the 4th member q is suspected). Then r sends a pull request to all of them to do data migration for the data-items that q was responsible for. If a peer receives a pull request, it retrieves all the data-items belong to the specified range and sends to the mentioned destination. When r receives the data from any of them, it only do an update of a particular data-item if $version_{received} > version_{existing}$ for an existing unlocked data-item or the data-item is absent in r . Thus, r may have stale value for a data-item temporarily, if it receives from e (which has old value) before a or m (holding up-to-date value), however eventually r will be consistent for the particular data-item. Though, this is not a perfect solution to achieve consistent and complete replica set for each data-item, as there might be non-overlapping replica sets, however this is a trade-off between cost and consistency.

Let us now analyze the lazy migration and simultaneous transaction of a data-item, k_i . Suppose, a , e and m are the replicas for k_i and after q is suspected r has become the member of the replica set. Now there can be 3 possible scenarios while lock is requested

4.2. TRANSACTIONAL DHT

for k_i by the transaction manager:

- r doesn't have k_i : r will vote for commit, so no inconsistency is introduced.
- r has a stale value of k_i : due to monotonicity of version number, there will be no inconsistency and r will vote for commit.
- r has up-to-date value of k_i : this is the ideal scenario, resulting in regular transaction.

So, in any case, lazy-migration of a data-item to the new member of the replica set doesn't create any conflict with simultaneous transaction of the same data-item. This is a low cost optimistic protocol to achieve completeness and higher consistency of replica sets.

4.2.2 Evaluation

This section evaluates the impact of Lazy-Data-Migration on the data level parameters, as described in Chapter 3. For this experimental study, a network of 100 peers has been bootstrapped. Correction-on-* principles are used as the sole maintenance strategy. As in Section 4.1.2, the statistical properties in [49] are used to simulate the underlying network. A consistent workload is created by injecting transactions, whereas transactions are modeled as a homogeneous Poisson process with $\lambda = 1$ per second. The workload is kept in such way that without churn the data-layer operates perfectly, i.e., no failed transactions and inconsistencies. A transaction is designed as: it reads k_i and updates k_j , here update means k_j is read and if it exists the value of k_j is incremented by 1. The replication factor is kept as 4. As in Section 4.1.2 churn events (join and crash with equal probability) are modeled as a homogeneous Poisson process with rate λ events/sec, whereas $\lambda = 2 * C/5$ for a churn of $C\%$ on a network of 100 peers. In order to understand the impact of Lazy-Data-Migration protocol, all data level parameters are measured once with lazy-migration and once without lazy-migration for each value of churn. The 2 sets of result for each parameter are plotted together so that they can be easily compared. Figure 4.5-4.8 show 4 data-level parameters for the 2 sets of result.

As we can see in Figure 4.5, increasing churn leads to more failure of data layer operations. However, lazy migration doesn't create any conflict with transactions, on the other side shows improvement as the availability and consistency of replicas are improved. As expected, the main impact of lazy migration is clearly visible in Figure 4.7 and Figure 4.8, as the integration of this simple, low-cost and optimistic approach leads to less key loss and inconsistent replicas. Figure 4.6 follows the trends of Figure 4.5, as explained before, more churn leads to more failed transactions i.e. more updates are lost.

4.2.3 Related Work

The most relevant work on the performance evaluation or dependability analysis of data layer operations of SON under churn is found in [56]. This work evaluates the frequency of inconsistent lookups, overlapping responsibilities and unavailability of keys in Chord [10],[11] resulting from unreliable failure detectors and churn. There are two other theoretical works on churn in Chord: [57, 58]. In [57] master-equation-based approach is

CHAPTER 4. FAILURE DETECTION AND REPLICA MANAGEMENT

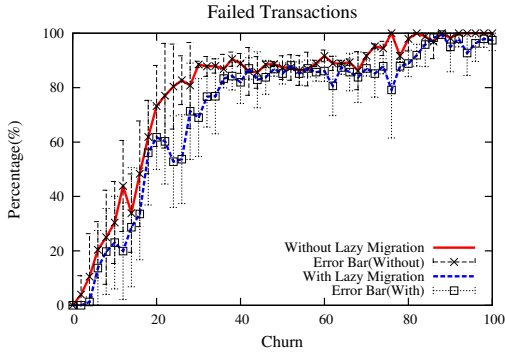


Figure 4.5: Data Level Parameter: % of failed transactions

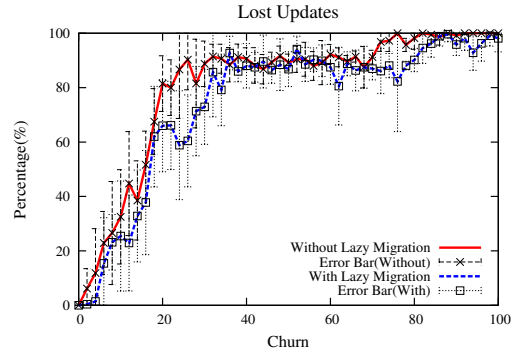


Figure 4.6: Data Level Parameter: % of lost updates

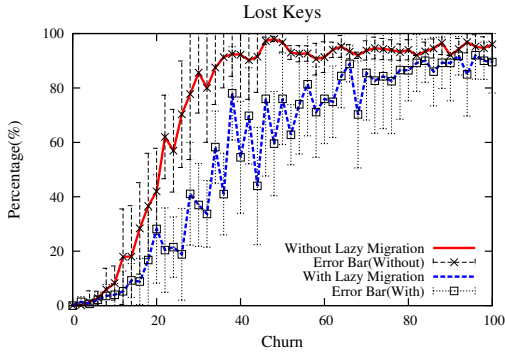


Figure 4.7: Data Level Parameter: % of lost keys

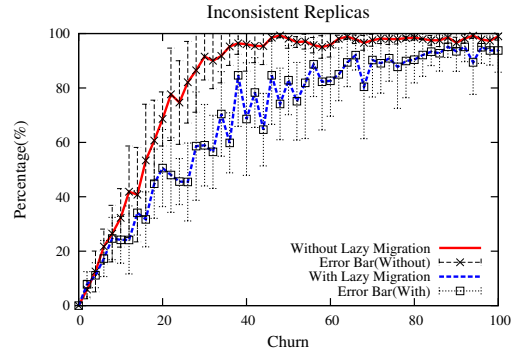


Figure 4.8: Data Level Parameter: % of inconsistent replicas

used to predict the performance and consistency of lookups under churn. A fluid model of Chord under churn is proposed in [58]. Our optimistic low-cost lazy data migration shows improved data-layer operations under churn using only correction-on-* as the sole maintenance, thus can be used as a complementary of existing works.

Chapter 5

Investigation about Churn

For a peer-to-peer network, node turnover (nodes failing and being replaced by new correct nodes) is the most usual scenario. Though, in most existing applications churn remains under a certain limit, as per studies [29, 30, 31] systems with low/average churn face high peaks. Consider the scenario of a *Structured Overlay Network (SON)* running on mobile phones or on an ad hoc network. Such a dynamically changing underlying network has still not been used because the environment is considered to be too inhospitable. A major goal of this work is to explore how to build systems that are able to survive and give useful functionality for such environments. This knowledge can improve system design with enhanced self-managing properties, while opening new vistas for applications - it is also one of the future scenarios that we investigate.

We construct a SON that is able to survive extremely high levels of churn, and when churn returns to a low value, the functionality originally available at the low value will again be available. We design our SON using the self-management principles necessary to make it reversible. To our knowledge, no previous SON provides reversibility for the high values of churn we investigate. We examine the reasons why and we add the maintenance principles necessary to make it reversible. We demonstrate reversibility through simulation using realistic network conditions and churn varying over a large range.

5.1 Reversibility

In this thesis, we introduce the concept of *Reversibility*. The term “reversible” implies that the system reverses back to its original state when a stress is withdrawn. We extend our SON to make it reversible, i.e., its functionality is a property of the current network conditions, and is not affected if the network has had failures in the past. This chapter focuses on one property of the network, namely *Churn*, which is the rate of node turnover, i.e., nodes failing and being replaced by new correct nodes. We assume that churn varies over time, and that the average number of correct nodes at any instant is constant. At the system level, the overall functionality depends only on the current intensity of churn and not on the history of this value. High churn will cause certain functionalities to disappear and when churn decreases, they come back. Therefore there is an analogy between high

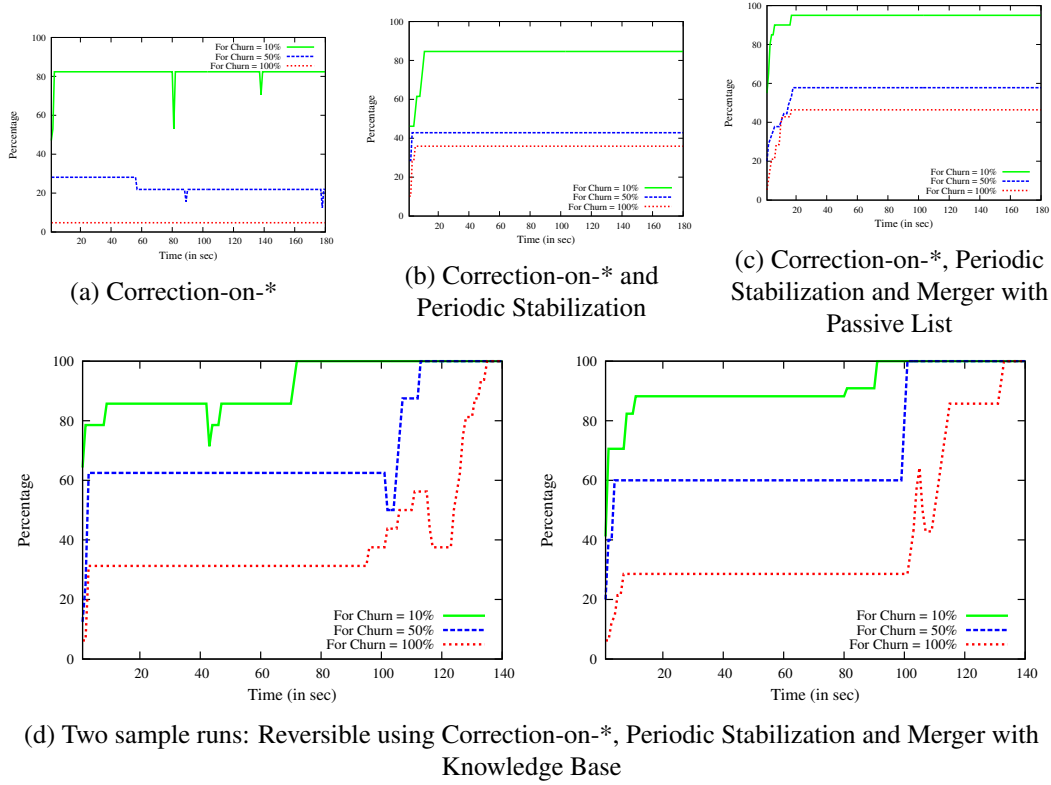


Figure 5.1: % of nodes on core ring as a function of time (in sec) after withdrawing churn to assess reversibility. Figure 5.1a, 5.1b and 5.1c are not reversible (nodes on core ring never converges to 100%). Figure 5.1d using Knowledge Base is reversible.

churn and temporary failures, at the system level, even though at the individual node level there are only permanent failures.

5.2 Evaluation of Reversibility

In order to ensure reversibility after experiencing continuous high churn, *Knowledge base (KB)* is essential. This is verified through our experimental results presented in Figure 5.1. To our knowledge no existing work demonstrates reversible SON for high churn by using one or more maintenance principles. We conduct step-wise assessment of reversibility against extremely high churn. At each step, we integrate a principle and evaluate to analyze the enhancements and lacking. Finally, we present a reversible SON, as shown in Figure 5.1d and explain what maintenance principles are needed to achieve reversibility.

In our evaluations, we use the Beernet [13] system. This is a straightforward SON that supports all the maintenance principles. It is an example of the reference architecture. For assessment of reversibility we use a SON of 100 peers. All experiments are done in Mozart-Oz 2.0 [59] in a simulated environment. To simulate the underlying network, the

5.2. EVALUATION OF REVERSIBILITY

end-to-end delays are set based on the empirical distribution of minimum RTT provided in [49]. The distribution represents significant geographic diversity, so we can say that the simulated SON is geo-distributed. We have defined churn as described in Chapter 3: percentage of nodes turnover per time unit (second in this work). If we assume equal probability of join/leave event and a single event per time unit, then every other time unit, a node will leave and a new node will join the network, i.e., every other time unit the total number of peers will be the same, whereas only a single node has a changed identity. In the steady state of SON when all nodes are on core ring, we have injected 10%, 50% and 100% churn for 1 minute. The churn events are modeled as a *Homogeneous Poisson Process (HPP)* with λ events/sec, where $\lambda = 2 * C$ for $C\%$ churn on a network of 100 peers. After withdrawing churn, we observe healing capability of SON with time. We have used 3 values of churn to compare and study healing for low, medium and high churn. In order to quantify the effect of self-healing, we have used metric: percentage (%) of nodes on core ring, to measure the rigidity of the ring. In other terms, it finds out the maximal ring in the system and reports % of nodes on it. The ultimate goal is to have the metric converge to 100%. A fixed workload is used in an experiment by injecting transactions, modeled as an HPP with $\lambda = 1$ transaction/sec. A transaction reads one key and updates another one.

5.2.1 Correction-On-Change and Correction-On-Use

As Figure 5.1a shows correction-on-* principles fail to achieve reversibility even for the lowest intensity (10%) of churn used in our experiments. Correction mechanisms allow the pointers to get fixed as soon as a failure, leave or join is detected, rather than waiting for a periodic check. Such rapid response reduces the probability of inconsistency and has more efficient bandwidth consumption. However, after withdrawing churn, the structure of the system remains almost the same. The reason for this is as follows. As healing is only triggered whenever a join/fail event occurs, so after churn is withdrawn there is no such event to continue the healing process. Also, under churn, a node based on its current state handles a new event, whereas the current state of the node might already be not optimized, for example, it might be on a branch and have missed an opportunity to reduce branch size. In such situation, handling high churn aggravates the structure, by unnecessarily pushing more and more nodes on branches. Although Beernet allows branches, but increasing branch size affects routing efficiency and increases the probability of creating isolated branches, thus inconsistencies under churn, as explained in [13]. So it is ideal and required to make branches shorter by bringing nodes on the ring and make the ring perfect whenever possible. But the branch-pruning algorithm is triggered only when a node joins, making it less effective. A node only heals whenever it senses an event based on its current state and later if a correction is required there is no option to trigger or propagate that, thus the damage of the structure remains as it is, at times even worsens as found in Figure 5.1a, once the churn events fade away. For churn value of 10% and 50%, we can see decrease of the number of nodes on core ring after withdrawing churn. The reason is the issuance of false suspicions by the failure detection, which push nodes on branches. Moreover, dependency on such non-perfect failure detection also affects a node's sensing capability, thus controlling how fast these principles will react. All these make the healing

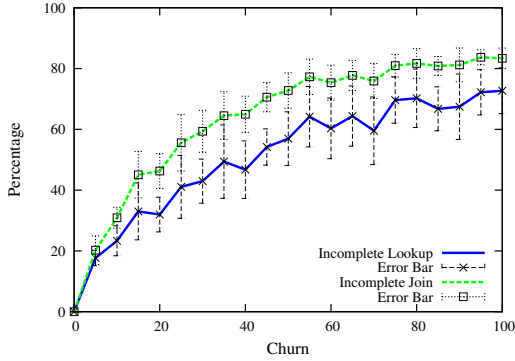


Figure 5.2: % of lookups and joins which remain incomplete after injection of churn for 1 minute

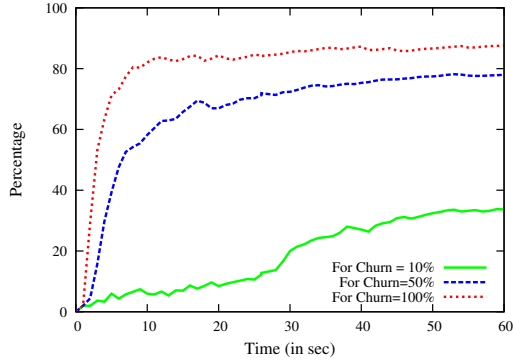


Figure 5.3: % of incomplete joins with time during injection of churn for 1 minute

capability, thus reversibility of these principles limited in the face of extremely high churn.

5.2.2 ReCircle (Periodic Stabilization and Merger with Passive List)

As mentioned in Section 3.2.3, ReCircle is designed using the principle of periodic stabilization. It has two parts: periodic maintenance and merger, triggered when the queue at a node becomes non-empty. Each node independently performs periodic maintenance by exchanging messages with its successor to maintain the local geometry. The periodic stabilization thus provides a shortsighted vision for each node about the current state of its immediate neighborhood. Moreover, a correction gets propagated in the identifier space during the subsequent rounds of periodic stabilization. Thus using this proactive mechanism eventually all local anomalies get fixed.

We can see significant improvements in Figure 5.1b, after integration of periodic stabilization; however still unable to achieve reversibility. The period used in our experiments for periodic stabilization is 1 second. Due to periodic maintenance, nodes on branches eventually become part of the core ring. However the healing is taken place only during first few seconds, and after that there is no change in the SON structure, as shown in Figure 5.1b.

As Figure 5.1c shows, the integration of merger with passive list does not show any significant improvement over the combined local healing. The reason is the existence of isolated nodes in the system (explained shortly). The nodes on the overlay has no reference to these nodes, i.e., no suspicion is issued. As a result, reactive merge attempt gains no success. The period used to dequeue the elements to generate *mlookups* at each node is 3 seconds and we have kept the *fanout* parameter as 1. Though the integration of this reactive global maintenance is unable to achieve reversibility, the knowledge is navigated further in the identifier space, triggering local maintenance during subsequent round of periodic stabilization, as evident from Figure 5.1c.

5.2. EVALUATION OF REVERSIBILITY

Why not reversible yet? In Figure 5.1c all nodes are still not on the core ring, i.e., the system is still not reversible. As our investigation shows, there are peers whose joining processes fail under churn. With the increment of churn more peers are unable to join the network. This is most likely a phase transition as discussed in Chapter 6, where continuous high churn injection makes the relaxed ring unstable that do not allow new peers. The first step of joining a SON is to do a *lookup* for successor and after receiving a response the new peer becomes part of the SON. For the join to fail, there are two possible reasons: i) the lookup request is lost while routing, ii) the successor has failed after receiving the lookup request; in both cases the new peer keeps on waiting for response. If we ignore the processing time at successor, then the probability of join failure is proportional to lookup failure. Figure 5.2 shows % of incomplete lookups and joins for different values of churn. We have used the same experimental setup of 100 nodes and injected increasing churn for 1 minute. Simultaneously, lookup requests are created using a HPP with $\lambda = 100$ request-sec. After waiting for another 5 sec, we report % of lookups which remains unanswered, also Figure 5.2 shows % of join requests which are not yet complete. As churn increases more and more join requests remain pending. We also present the accumulation of pending join requests with time in Figure 5.3, especially for high churn. As it is evident, with high churn, the join request of the new peer is lost.

5.2.3 Knowledge Base for Each Node

As shown in Figure 5.1d, after integration of knowledge base the system has achieved reversibility. We have conducted 50 independent runs for each value of churn, all of which have converged. This gives an upper bound on non-reversibility. Out of the 50 samples collected, we present two representative runs in Figure 5.1d for each value of churn. As the churn intensity increases, more and more join requests remain pending, which isolates these peers. We have described in Section 3.2.4 two alternative solutions for this problem. Out of those we have chosen the simpler one for our experiments: providing a valid join reference and re-triggering the join request. If a node is unable to join with its current join reference within 100 seconds, then it requests a new join reference from the application layer and triggers new join request with that. This is a tunable parameter, which we will refer as *Join Timeout*, we have chosen a conservative value of 100 seconds for this parameter to avoid triggering of unnecessary repeated join requests. This parameter can be adapted based on the operating environment and RTT distribution of the underlying network, which we leave as future work. Along with this, we have used the proactive manner of triggering the merger using KB. In some runs we have observed partition of the system after the isolated nodes complete their join procedures. In order to merge such partitions proactive global maintenance using KB is required. As we can see in Figure 5.1d, for both samples, the damage of the structure caused by high churn is completely healed; thus the system is reversible.

The healing follows a staircase trend. As periodic stabilization is done in each second, after withdrawing churn during first couple of rounds all the corrections are performed among the nodes of the overlay. The remaining nodes are the isolated ones, whose join request is not complete yet. These nodes after 100 seconds retrieve a new join reference

and trigger a new join attempt. So we get a stair of larger width due to this waiting period, after which all nodes converge to a perfect ring.

5.2.4 Discussion

This section has presented construction of a healing mechanism, which can make the system reversible. For this purpose we have integrated different principles and assessed the enhancement of healing capability after each extension. Also, we have analyzed the shortcomings of each principle. This experimental study shows that, in order to have an effective self-healing behavior, the healing mechanism should have the following properties:

- It should react to an event immediately, so that no inconsistency is introduced;
- It should have a periodic maintenance mechanism to check for correction opportunities. The frequency of this can be tuned based on available resource and requirement to avoid unnecessary bandwidth consumption;
- Proactive global maintenance is essential in order to achieve reversibility against inhospitable environments caused by extremely high churn. For this purpose, each participant should keep on acquiring knowledge, share this with neighbors and also navigate further. This allows each node to have an approximation of the global state. The Knowledge can be used to trigger the global maintenance as required, also can prove to be invaluable in order to face an inhospitable environment.

5.3 Evaluation of High-Level Properties

We evaluate how functionality (e.g., transactions, storage, replication) decreases as churn increases. Again, we use the Beernet [13] system, which provides strong consistency with transactions, key/value storage, and replication. We have analyzed the following metrics:

- Damage to the ring topology in terms of % of nodes on core ring;
- Topology recovery/healing time and cost;
- Data level parameters: % of failed transactions and % of lost keys, % of inconsistent replicas, % of lost updates.

We use similar experimental setup as described in Section 5.2. We bootstrap a SON of 100 peers. Correction-on-*, periodic stabilization and merger with Knowledge Base are used as part of maintenance. The underlying network is simulated based on the empirical distribution of minimum RTT provided in [49]. We have used a stream of transactions to create workload, where transactions are modeled as a HPP with $\lambda = 1$ transaction/sec. A transaction reads one key and updates another one. During steady state of the SON, churn of increasing intensity is injected. For each value of churn, Beernet++ experiences join/leave events for 1 minute. The churn events are also modeled as a HPP, as explained before. We have used mean value of 10 independent runs for every 5% increase of churn.

5.3. EVALUATION OF HIGH-LEVEL PROPERTIES

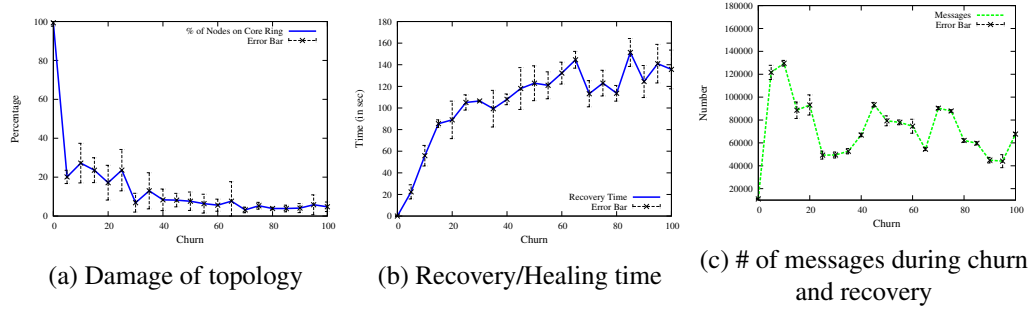


Figure 5.4: Properties for increasing churn after injecting a particular churn value for 1 minute

5.3.1 Damage and Recovery of Ring Topology

After churn is withdrawn; we present the snapshot of the ring topology (in terms of % of nodes on core ring) of the overlay in Figure 5.4a. This gives an idea the damage done to the topology due to increasing churn. We also measure the time (in seconds) required for the existing nodes to organize into a perfect ring topology once churn is withdrawn. Figure 5.4b presents the recovery time or time to heal the damage (as shown in Figure 5.4a) of ring topology for increasing churn. As expected, it requires more time for the SON to regain/restore its topology with increasing churn. We also present the number of messages generated for each value of churn in Figure 5.4c, which provides an approximation of bandwidth consumption for the self-healing of Beernet++. We find in Figure 5.4c, the number of messages is higher for lower churn, the reason behind this is, for lower churn there are less number of isolated nodes, i.e., almost all the nodes are part of the SON. These nodes periodically issue maintenance messages, whereas an isolated node doesn't have reference of any other node, so no periodic message is exchanged. This, in fact, reflects the impact of costly periodic stabilization. In order to make the overall maintenance efficient and scalable, it is essential to adapt with the operating environment, which we keep as future work.

5.3.2 Data Level Parameters

To understand how data-layer performs during a challenging environment like extremely high churn we present percentage of failed transactions in Figure 5.5. As expected, as the intensity of churn increases, almost all transactions abort. After churn is withdrawn we do a read transaction to read all data items in order to understand the impact of the harsh environment on the data storage and report percentage of lost keys in Figure 5.6. After churn is beyond 30%, almost all keys are lost. From this result, we can conclude that in order to survive phase transitions due to an adverse operating condition, the application design needs to take into these issues into consideration. The remaining two parameters, namely fraction of lost updates and inconsistent replicas, follow the trends of percentage of failed transactions and lost keys respectively.

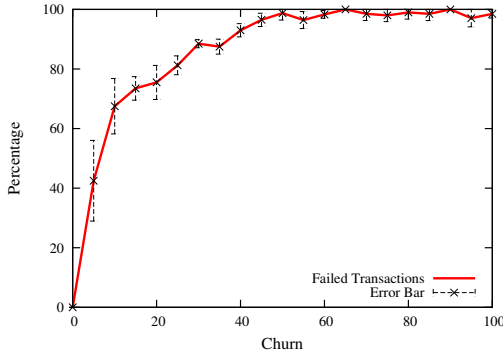


Figure 5.5: % of failed transactions

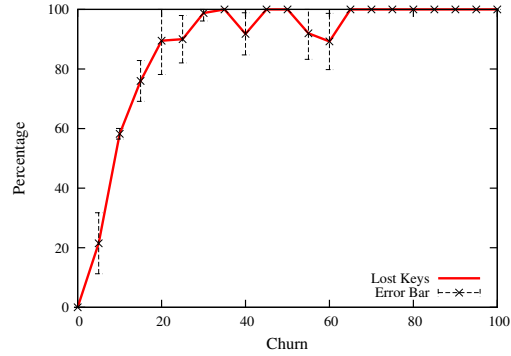


Figure 5.6: % of lost keys

5.4 Related Work

The work in [60] mainly focuses on routing level correctness/consistency and improved performance of a SON. The evaluation of MSPastry (a new implementation of Pastry[61]) is conducted by varying environmental parameters like network topology, node session times, link loss rates, and amount of application traffic. This work validates proposed techniques for improved routing performance and dependability, in the face of high churn. Another work on lookup consistency, [56] evaluates the frequency of inconsistent lookups, overlapping responsibilities and unavailability of keys in Chord [10],[11] resulting from unreliable failure detectors and churn. Krishnamurthy et al. in [58] use fluid model approach to do theoretical analysis of Chord [11] under churn. They present the functional form of the probability of network disconnection and the fraction of incorrect pointers (successor and fingers). In [57] master-equation-based approach is used to predict the performance and consistency of lookups under churn. Also, in their continuing analytical study in [62] use the analytical tool based on master-equation approach of physics to do comparative analysis of periodic stabilization and correction-on-change maintenance principles under churn. Another analytical work in [63] establishes a lower bound on the maintenance rate of a SON under churn in order to remain connected. In [64] and [65], El-Ansary et al. use physics-inspired analytical approach to analyze performance of large scale distributed systems and also investigate about intensive variables (i.e., variables which are independent of system size) related to self-organization and self-repair. They apply this methodology for Chord and propose an intensive variable to describe the characteristic behavior of the overlay. Apel et al. [66] describes design decisions such as self-tuning mechanisms based on software-engineering principles for self-organization/self-adaptation of overlay networks. The analytical framework presented in [67] can be used to characterize the routing performance of SON under churn. Our empirical study can be used as complementary of these analytical works.

Chapter 6

Phase Transitions

A *Structured Overlay Network (SON)* provides significant functionality to the applications running on top, e.g., transactions over key/value store. However, as the inhospitability of the operating environment of a SON continues to increase, it will no longer be able to provide such functionalities. Thus, applications that rely on transactions will no longer be able to use them. We would like these applications to continue running nevertheless, with predictable behavior even though functionality will be less. Ideally, this should be done in a manner that works even for operating environments which are extremely inhospitable. The SON can therefore not be relied on to do additional computation to determine its level of functionality. Under this constraint, is it possible for the SON to give useful information to the application?

In this chapter, we consider the environment inhospitability to be measured by the *Churn* parameter, i.e., the rate of node turnover (nodes failing and being replaced by new correct nodes). In order to describe the behavior of a SON, we introduce the concept of *Phase* of the system. The SON's phase is a qualitative description of the structure of the SON. For example, at low churn, the SON has a mostly fixed structure, and at high churn, the SON can decompose into small rings or single nodes. For low churn, the SON has full functionality, and at high churn, the SON has reduced functionality. The phase of the SON is not a global property, but is observed separately at each node, and can be different for different nodes. No global synchronization and no extra computation is required to compute the phase; it is a direct consequence of the observed structure of the SON at each node. Thus, the phase inferred at each node correlates with SON functionality and can allow the application running on that node to modify its behavior depending on the functionality available for the current churn value. In this chapter, we experimentally demonstrate *Reversible Phase Transitions*: the nodes of the system change phase as the churn is varied. We note that the concept of phase and phase transition are analogous to phase in physical systems; we have chosen the term phase for this very reason. Phase transitions is the consequence of having a reversible system, Beernet++ (see outcome of Chapter 5), i.e., the SON we have constructed is able to survive extremely high levels of churn and reversible.

6.1 Definition of Phase, Phase Transition and Critical Point

A *Phase* is a subset of a system for which the qualitative properties are essentially the same. For this definition we consider a system as an aggregate entity composed of a large number of interacting parts. Different parts can be in different phases, depending on the local environment observed by the part. Boundaries between phases in a system can be either sharp or diffuse. A *Phase Transition* occurs when a significant fraction of a system's parts changes phase. This can happen if the local environment changes at many parts. A *Critical Point* occurs when more than one phase exists simultaneously in significant fractions of a system. Phases are observed in many large systems. They are well-understood in physical systems consisting of large numbers of atoms or molecules (e.g., macroscopic amounts of water, in solid, liquid or gaseous phases) [68], but they can also be observed in computing systems. In this chapter, we investigate phases and phase transitions in a SON due to churn, where each part is a node of the SON.

We observe three qualitative structures in Beernet++: core ring, branches, isolated ringlets. We call each of these structures a phase. As we will see, there is a close analogy between these three phases and the solid, liquid, and gaseous phases in physical matter (e.g., water). We define these three phases in the context of our SON:

- **Solid:** The *solid* state of a matter is characterized by structural rigidity, where atoms or molecules are bound to each other in a fixed structure. In case of SONs, a stable ring with a stable finger table, where all participants are self-organized in a predefined structure, can be termed as a solid state of the SON. During solid state of Beernet, each node has fixed neighbors, all nodes are organized into a stable ring and there are no branches in the system.
- **Liquid:** A thermodynamic system is in the liquid state where molecules are bound tightly but not rigidly. In case of SON, with the increase of churn, the quality of the network decreases, more nodes move on branches, thus being less strongly connected, as in liquid. To be more precise, when there are non-zero nodes on branches, we can say that Beernet is in the liquid state, since each node does not have a fixed set of neighbors. There can be two forms of liquid state for Beernet: first, branches around a core ring (solid core surrounded by liquid), and second, branches with no core ring (only liquid, no solid core). In the second case, the core ring consists of a single node.
- **Gaseous:** The gaseous state of matter is made up of individual molecules that are separated from each other. When Beernet experiences high churn, new nodes are unable to complete a join, and at some point the ring is completely dissolved, where no node is able to join the ring anymore, resulting in isolation of all nodes. This state of the system can be termed as the gaseous state of a SON.

Phase is a property of each node. In SON, the phase of a node is clearly determinable at that node: there are three mutually exclusive situations depending on neighbor behavior

6.2. OBSERVATION OF PHASE TRANSITIONS

(no neighbors, neighbors on branch, neighbors on core ring). Also, each node changes its phase independently. The current phase and phase transition at each node can be determined with high confidence, without any global synchronization. However, at system level, phase transitions happen when there are changes in the phases of the majority of nodes. When external conditions change, each node changes phase. If that happens to many nodes, we have a phase transition at system level. It is not possible to determine at a single node, a phase transition at system level, because this would require a global algorithm.

The phase of each node has a direct co-relation with the overall properties (e.g., routing, availability of keys) of the system. The routing guarantee offered by the system is $O(\log_k N + b)$, where N is the total number of nodes and b is the branch size (i.e., the number of nodes on branches) of the system. So, the more nodes change to liquid phase, the routing will degrade with that. Also, with the increased branches in the system, the probability of introducing unavailability of keys under churn, also increases due to isolation of branches (as discussed in [13]).

6.2 Observation of Phase Transitions

We show experimentally the existence of phase transitions in Beernet++, as the churn intensity varies. For our experimental study, we have used a network size of 100 peers. To simulate the underlying network, the end-to-end delays are set based on the empirical distribution of minimum RTT provided in [49]. The churn events (join and failure) are modeled as a homogeneous Poisson process with $\lambda = 2 * C$ events/sec, for $C\%$ of churn and 100 peers. We have measured following 3 parameters for all figures in this section:

- Percentage (%) of nodes in solid state: nodes which form a ring, we collect these nodes by doing a routing using only the successor pointer of each node;
- % of nodes in liquid state: nodes which are on branches rooted at nodes on the ring (collected by using the routing mechanism of Beernet as explained in [13]) or nodes which do not form a ring but can be reached by routing;
- % of nodes in gaseous state: isolated nodes whose join requests are still pending.

6.2.1 Increasing Churn with Time

In this experiment we study phase transitions of Beernet++ under increasing churn and reverse transitions when churn is removed. We start with 5% churn and increase the intensity by 5% every 5 seconds for 5 minutes. As we reach churn value of 100, we keep on injecting 100% churn for the rest of the period. Every 5-second we take a snapshot of the system, where we measure the parameters described. After churn is withdrawn, we let the system running for another 4 minutes to do self-healing and every-5 second take the measurements. We have used mean value of the 3 parameters for 20 independent runs. Figure 6.1a and Figure 6.1b show the state of the system during injection and withdrawal of churn respectively.

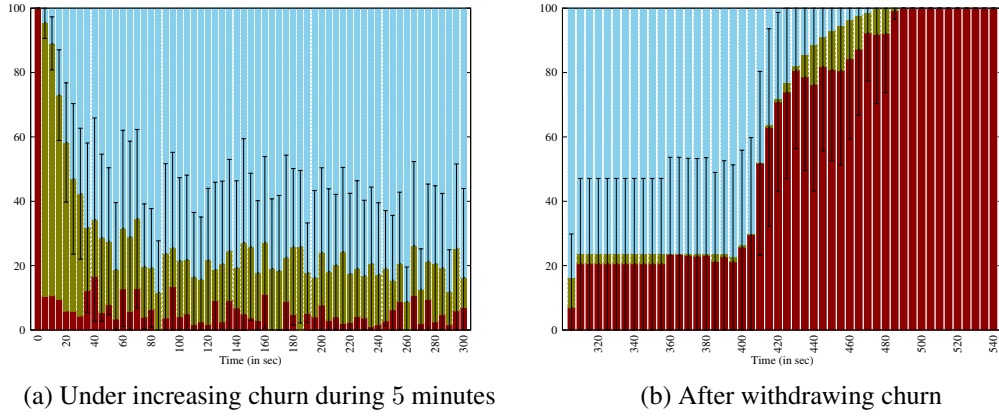


Figure 6.1: Phase Transitions in Beernet++: red, green and blue (dark, gray and light-gray in B/W) areas correspond to % of nodes on ring, branches and isolation respectively

Only error bars for gaseous state are used in Figure 6.1a. The red area of each bar corresponds to % of nodes which form a ring, thus in solid state. At time 0, i.e., starting of the experiment with no churn, all nodes are organized into a perfect ring. As churn is increased nodes start moving on braches, the green area of each bar, these are the nodes, which are in liquid state. The result shown is an average of the parameter values for several samples and each sample go through a phase transition at different instant of time. However, still we can figure out some trends. For example, from our observation, around 30% of churn mostly the ring gets dissolved, as there are no nodes on the ring, however there is sill connectivity among nodes. Finally the blue area of each bar represents the nodes that are waiting for the response of their join requests, which keeps on increasing with churn. The transition from liquid to gaseous state is periodical as can be analyzed from Figure 6.1a. If we combine the height of the blue bars with corresponding error bars, we can say that around 85, 170 and 260 seconds most samples go through transitions.

We have not identified the solid to liquid transition yet. As in Figure 6.1a, we can see a sharp fall of % of nodes on core ring from 0 to 5 sec, so the transition happens between 0 and 5% of churn. In order to analyze this transition we have zoomed in this area. For this experiment, during steady state of SON, we start with 1% of churn and every 5-second we increase churn intensity by 1% till we reach 5% of churn. Also a snapshot is taken every 5 second as before. Figure 6.2 shows the result. We have used mean values of 20 samples for the 3 parameters described above and only error bars for the solid state are shown. We can say that for churn value of 2, during 5 to 10 seconds, a transition happens for most samples.

Figure 6.1b shows the recovery of the SON after churn is withdrawn (i.e., 6 – 9 minutes of our experiment). For this figure we have used error bars for solid state only. Here starting with all isolated nodes, i.e., from gaseous state, the SON move to transient liquid state, where connections are built among nodes. We can see a period of about 100 seconds, during which the % of isolated nodes remains same. The reason behind this is the *Join Timeout* parameter, as described in Section 5.2.3, which is set as 100 sec. As the last churn value used is of 100%, according to our definition of churn, all nodes in the SON are

6.2. OBSERVATION OF PHASE TRANSITIONS

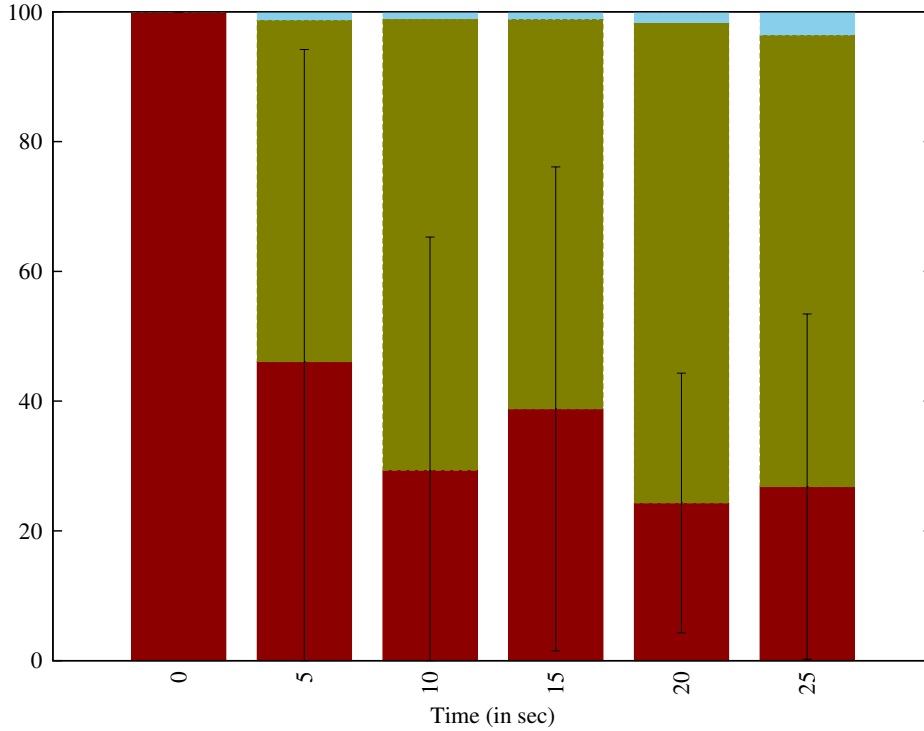


Figure 6.2: Phase Transitions in Beernet++ under low churn (0% to 5%): red, green and blue (dark, gray and light-gray in B/W) areas correspond to % of nodes on ring, branches and isolation respectively

replaced by new ones during a second. So, almost all nodes start their waiting period to receive a response around same time and get a timeout around 400 sec. As explained the transition from gaseous state is controlled by this tunable parameter. Finally all nodes are self-organized into a perfect ring, solid state of SON within 425 to 490 sec.

6.2.2 Continuous Moderate Churn

In this experiment we seek answer to the question: whether the entropy of a SON can be increased by continuous injection of low/moderate churn so that a phase transition is triggered. In order to investigate about this, we start with same experimental setup described before, but this time instead of increasing churn with time, we inject same value of churn. For this experiment we have chosen churn value of 30. From Section 6.2.1, we have observed around churn intensity of 30, as per our definition of churn and described experimental setup, there is a critical point. Around this point the ring structure starts fading, thus a phase transition happens. During steady state of SON, we start injecting churn value of 30 and every 5 second we measure 3 parameters (described at the beginning of this section) until we reach a gaseous state (i.e., all nodes are isolated). Then churn is withdrawn and we let the SON do self-healing until all nodes are on the core ring. Figure 6.3a and Figure 6.3b

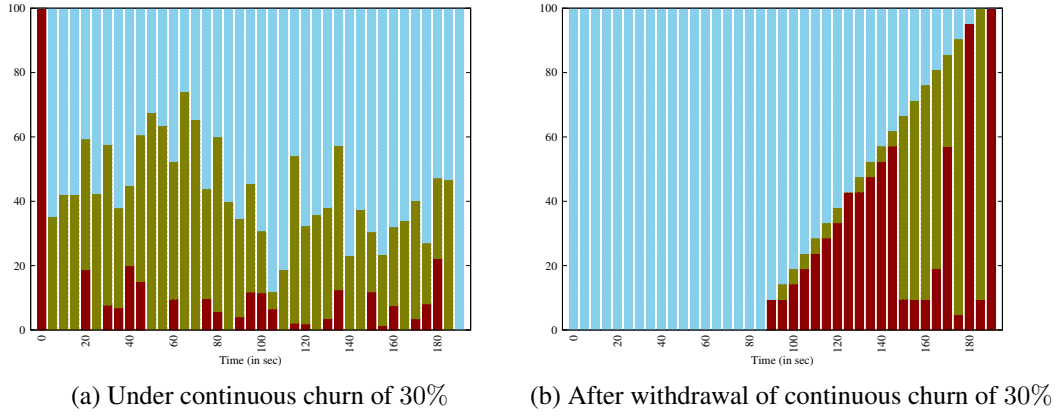


Figure 6.3: Phase Transitions in Beernet++: red, green and blue (dark, gray and light-gray in B/W) areas correspond to % of nodes on ring, branches and isolation respectively

show the result of a single run. We present the outcome of a single run in order to have explicit transitions. As in a thermodynamic system, it takes longer time to reach a gaseous state. Also as shown in Figure 6.3a, we observe that the SON quickly reaches the liquid state and there is hardly any ring structure kept throughout the experiment. The reverse transition shown in Figure 6.3b follows the same pattern due to same reason as described in Section 6.2.1.

6.2.3 Gradual Increase and Decrease of Churn

Till now, we have withdrawn churn completely; what behavior the system exhibits if the intensity of churn is gradually decreased? In this experiment we investigate about this. For this experiment, we use same experimental setup and parameters as already mentioned. During steady state of SON, we start injecting 5% of churn and increase the intensity of churn every 5-second until the system reaches a gaseous phase. Then we gradually decrease churn by 5% every 5 second until it reaches 0. We measure the parameters every 5 second throughout the experiment. The behavior is shown in Figure 6.4 for a single run. The ring structure starts fading away around 30 sec, which corresponds to 30% churn, justifies our deduction of churn intensity of 30 as one of the critical points. At 75th sec, we can see a gaseous phase. So, starting from 76th sec, the churn intensity is decreased gradually, for example the bar at 80 shows the snapshot of the system after injecting 70% of churn for 5 sec. As shown in Figure 6.4, during gradual decrease of churn intensity, there is increasing connectedness among nodes, followed by organization into ring structure, evidential of reversible phase transitions in Beernet++ due to increasing and decreasing churn intensity. This provides experimental justification of the conjectured phase transitions for relaxed ring SON in [69].

6.3. RELATED WORK

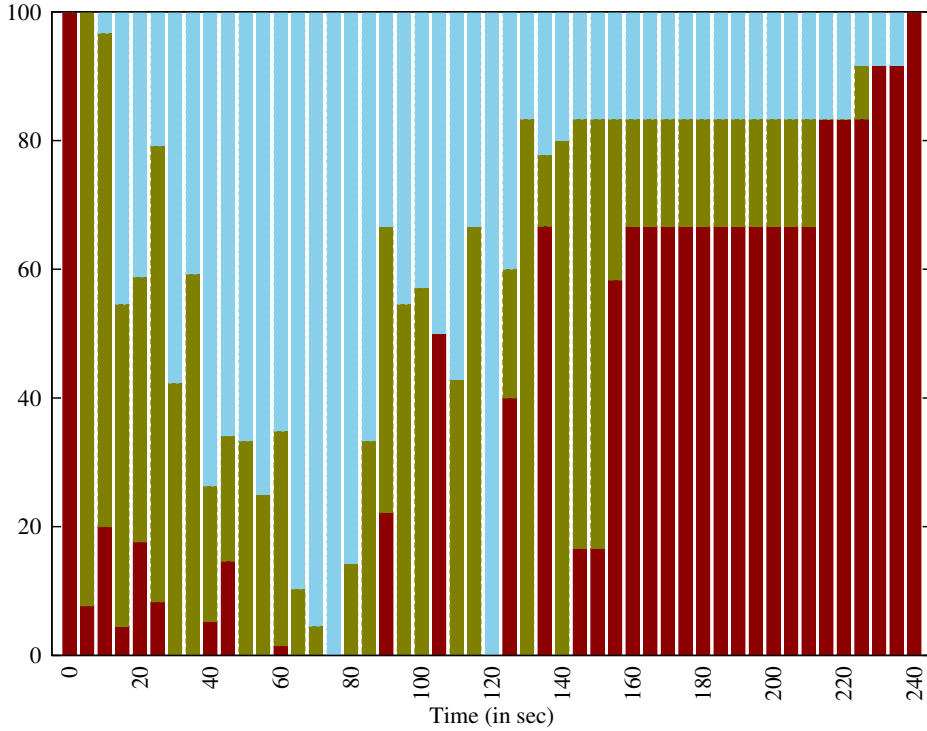


Figure 6.4: Phase Transitions in Beernet++ due to increasing and decreasing churn: red, green and blue (dark, gray and light-gray in B/W) areas correspond to % of nodes on ring, branches and isolation respectively

6.3 Related Work

Diligent search has failed to uncover any empirical work on phase transitions in structured peer-to-peer network, however we have found one analytical work [44]. The result of this study, carried out for Chord in [44], shows a critical point in parameter space at which the system with high probability breaks down, i.e., efficient routing becomes impossible. Such phase transitions happen due to high churn and large link delays, resulting in a finite fraction of the connections to be always incorrect/dead. We have come across works on phase transitions in other networked systems during our search. For example, the analytical work in [70] is based on theory of criticality and complex systems, which studies and applies the phase transitions phenomenon for unstructured peer-to-peer networks. In [71], Scholtes et al. present distributed monitoring and adaptation schemes of macroscopic statistical network parameters using power law networks. Such adaptation of critical parameters can be termed as phase transitions. For phase transitions in other topologies of network; for example the small-world model [72, 73, 74, 75, 76, 77], several authors have studied an Ising model. There are several studies [78, 79, 80] on phase transitions in models of Internet traffic. The outcome of these works can be applied in the context of SONs.

6.4 Discussion

In this chapter we have experimentally demonstrated reversible phase transitions in Beer-net++ for increasing/decreasing churn and continuous injection of moderate churn. Such phase transitions are consequence of having a reversible system. Other SONs of our representative design class (as described in Chapter 2) may be extended in a similar way to achieve reversibility. Also, we have analyzed the critical points observed in our experiments. The knowledge about critical points in the operating space of a SON can provide useful information to the application layer and administrator in order to initiate adaptation measures based on application layer semantics. It will also allow the application to give meaningful notifications to the users in terms of available functionalities.

Chapter 7

Investigation about Network Partitioning

In this chapter, we investigate a particular operating environment of *Structured Overlay Networks (SONs)*, *Network Partition/Merge*. During the partition of underlying network, the nodes of a SON are divided into multiple disjoint sets, where a node can communicate with the nodes of its own set, but is unable to contact the nodes in the other sets. Any long-running large distributed system is bound to come across network partitions during its execution. Consider the scenario of a SON running on mobile phones or on an ad hoc network, which has high node mobility and intermittent connectivity, and undergo frequent changes in network topology. In such a dynamically changing environment network partition can be a frequent event. Due to self-* properties (mostly partial), most of the ring-based SONs are expected to provide partition-tolerance by forming separate overlay(s) in each partition. However, once the partition ceases (we will term this event as *Network Merge*), the system should reverse back, i.e., merger of multiple overlays, resulted from endurance during partition. In this chapter, We evaluate partition-tolerance and merging (as the network partition ceases) capability of existing maintenance strategies, namely Correction-on-Change, Correction-on-Use, Periodic Stabilization, and Ring Merge. We identify the necessary and sufficient maintenance strategies to ensure partition tolerance for any scenario of network partitions. By means of simulations, we demonstrate reversibility, once network partition ceases, for overlay networks with high levels of partition and we make general conclusions about the ability of the maintenance strategies to achieve reversibility. In this chapter, we only consider scenarios where no churn is induced during partition, which corresponds to network partition of short duration.

7.1 Types of Partition

There can be 2 different types of partition of the overlay that may arise as a result of the underlying network partition. This differentiation happens due to the *locality-awareness* of the mapping function F_p , which associates peers with a unique virtual identifier from the identifier space (Section 2.1). If F_p is locality-aware, then logically adjacent nodes (on

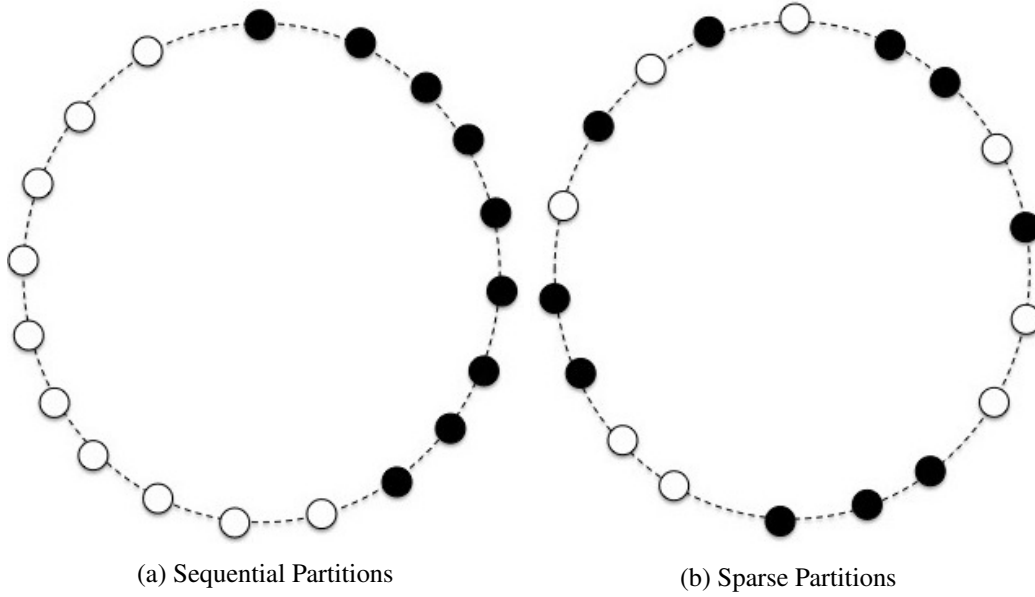


Figure 7.1: Two different types of partition scenarios: white and black nodes belong to two different partitions

the overlay) are also physically close; e.g. for DKS F_p is order-preserving to ensure that nodes in same organizational domain are logically close on the overlay. As a result, an underlying network partition usually divides such overlay into P contiguous regions (here P is the number of partitions). We will refer this particular kind of partitioning as *Sequential Partitions*, which emphasizes locality. Figure 7.1a shows an example of sequential partitions. On the other hand, if F_p is not locality-aware or some random function then logically adjacent nodes can be very far apart physically, e.g. for Chord, F_p is a uniform hash function, which uniformly distributes nodes on the identifier space. For such mapping, an underlying network partition might cause adjacent nodes on the overlay to be in different partitions. We will refer such partitioning as *Sparse Partitions*, which emphasizes non-locality. In Figure 7.1b, we can see sparse partitions, where adjacent nodes tend to be in different partitions.

7.2 Evaluation of Maintenance Strategies

We distinguish two different cases for a SON, corresponding to a network partition: 1) execution during network partition (i.e., partition tolerance) and 2) partition repair (i.e., achieving reversibility). Existing works do not analyze case 1 in depth, have only mentioned the particular partition scenarios for which a maintenance strategy is able to provide partition tolerance [13]. Shafaat et. al. [40] do not address case 1, their work is about merging of multiple ring overlays, i.e., case 2. We have identified the preconditions to ensure partition tolerance for any possible partition scenario. For case 2, there are several works on

7.2. EVALUATION OF MAINTENANCE STRATEGIES

Scenario	Local Maintenance (Correction-on-*, Periodic Stabilization)	Global Maintenance (Merger with Passive List/Knowledge Base)
Execution During Network Partition	Can Create separate rings in each partition but can get stuck	Merger with KB is Required to provide the best partition-tolerance; however Merger with passive list can fail to fulfill the requirement
Execution At Partition Repair	Combined reactive and proactive corrections is able to merge multiple overlays, even reacts quicker	Provides no improvement over the combined local strategies

Table 7.1: Self-healing achieved by Maintenance Strategies

ring merge algorithm [40], [81], but no work has been found that assesses partition repair capability of local maintenance principles and compares their performance with an explicit merger. We have analyzed ring merge capabilities of correction-on-* (with and without periodic stabilization) and shown that integration of an explicit merger gains no significant improvement if no churn is induced during partition. Table 7.1 summarizes our result. In order to have the best partition-tolerance during a network partition, global maintenance is required along with the local corrections. To repair partitions, combined reactive and proactive local corrections are enough to achieve complete self-healing, which even shows quicker response as the network partition ceases.

7.2.1 Execution During Network Partition

In order to ensure partition tolerance for any partition scenario, proactive global maintenance, i.e., the merger with KB is required along with local ones. During partition it is impossible to achieve both global consistency and availability as per the *CAP* theorem [82], however each ring should be consistent in itself. In our work we refer a SON to be partition tolerant (or survive a partition) iff i) the peers on each partition will form a separate overlay, ii) the system shows high availability i.e. every lookup must result in a response and iii) each overlay is consistent in itself. Both correction-on-* and periodic stabilization is able to provide partition tolerance as long as every peer is able to find a valid successor candidate in its successor list, as also identified in existing literature. We will analyze the partition tolerance capability of these two local maintenance strategies, when this condition does not hold, using the example scenarios in Figure 7.2.

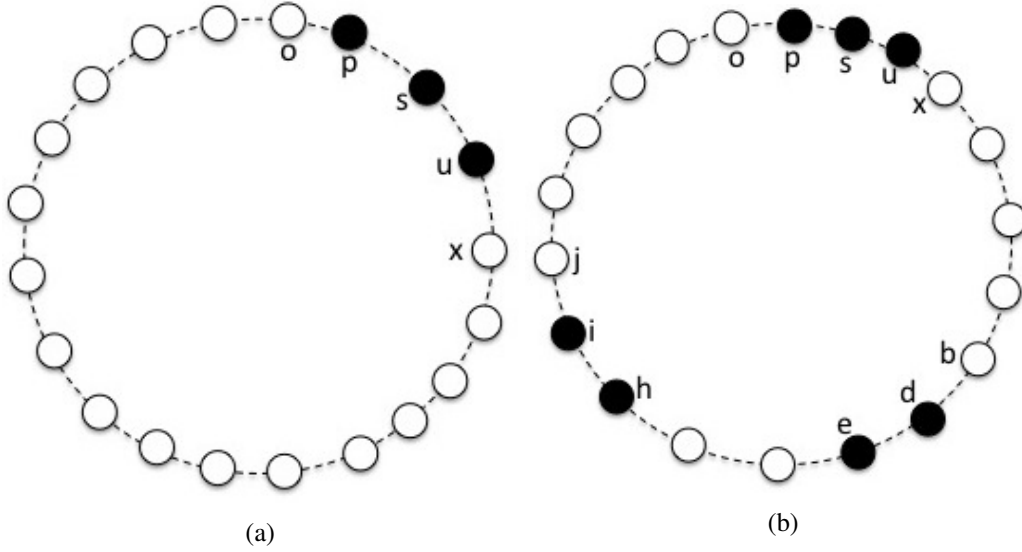


Figure 7.2: Two partition scenarios: white and black nodes belong to two different partitions; partition having black nodes have absence of more than $|succ_list| - 1$ consecutive peers (here, $|succ_list| = 4$).

Local Maintenance

As already mentioned, both correction-on-* and periodic stabilization are able to survive a network partition as long as no more than $|succ_list| - 1$ consecutive peers are absent from a partition. Comparing two different partition types, described before, sequential partitions are more prone to face scenarios, where this condition is not satisfied, than the sparse partitions.

Using only correction-on-* is insufficient to provide partition tolerance for the partition scenarios in Figure 7.2. When peer u suspects its current successor x due to network partition, it fails to find a valid successor in its successor list to trigger the failure recovery, as the next consecutive 5 peers are partitioned away (which is $> |succ_list| - 1 = 3$), thus sets its successor pointer as itself. In a similar way, peer i also fails to find its successor in Figure 7.2b. Peer p (and peer d in Figure 7.2b) suspects its current predecessor due to partition, but as nobody triggers the recovery mechanism, the responsibility of peer p (and peer d in Figure 7.2b) doesn't change, resulting in gap in the identifier space. The situation remains the same until the partition ceases. This introduces unavailability of keys in the range $(u, o]$ in Figure 7.2a and $(u, b]$, $(i, o]$ in Figure 7.2b for the partition holding black nodes. Also, the nodes fail to form an independent overlay in this partition as per the predefined structure or embedded graph. So, correction-on-* fails to survive these partition scenarios.

Using periodic stabilization improves availability, but multiple ring overlays are formed in the same partition. For Figure 7.2a, peer u unable to find a valid successor in its successor list after the partition, sets its successor pointer as itself. During next round of periodic

7.2. EVALUATION OF MAINTENANCE STRATEGIES

stabilization (suppose $round = t$), s asks u about its predecessor. On receiving this message, u sets s as its successor, as s is a better successor than itself. During $round = t + 1$, u asks s about its predecessor and comes to know about p . As p is a better successor for u , u sets its successor pointer as p , which also triggers change of responsibility of p . Thus, eventually p , s and u form a separate overlay in this partition and the gap in the range $(u, o]$ is healed. Though temporary unavailability of a certain range of keys do occur, it is possible to eventually overcome that, where peers on each partition form an independent overlay, which is consistent in itself and provides availability. However, for the partition scenario in Figure 7.2b, this conclusion does not hold entirely. As we can see in Figure 7.2b, there are two instances of minimum $|succ_list| - 1$ consecutive peers are absent from the partition holding black nodes. As in Figure 7.2a, peers p , s and u form a ring, so do peers d , e , h and i . So, there are two ring overlays formed in the same partition and remains the same, until the partition ceases. The reason is that periodic stabilization only does local healing around a node's immediate vicinity and lacks the mechanism to spread out the healing globally. So, for the partition scenario in Figure 7.2b, the healing of periodic stabilization falls short to satisfy the first condition of partition tolerance (as described before), where there are more than one overlays formed in a partition (and remains the same), though the nodes on the overlays are able to communicate with each other. As per our analysis, during a network partition, the number of overlays formed in the same partition is equal to the number of instances of minimum $|succ_list| - 1$ consecutive missing peers from that partition.

Global Maintenance

As described before, local maintenance can get stuck while providing partition-tolerance. For example, for the representative partition scenario in Figure 7.2b, periodic stabilization eventually organizes the nodes in the partition (holding the black nodes) into two different overlays. In order to merge these two overlays, we need an overlay merge algorithm, e.g. the merger of ReCircle. If any peer from one overlay is enqueued into the queue of a peer of another overlay, the merger will be triggered, thus eventually resulting into a single overlay in the partition (holding the black nodes). Using the passive list approach for this purpose fails to trigger the merging process, as nodes in the same partition are not suspected by each other. We have experimentally verified this result for the representative partition scenario in Figure 7.2b. However, triggering merger in a proactive manner using knowledge base is able to provide the desired outcome, as also validated via simulation. By exploiting the knowledge base at each node periodically (as described in Section 3.2.4), eventually the merger will be triggered as a result of enqueueing a valid peer of other overlay, with which communication can be established, thus resulting into a single overlay in the partition (holding the black nodes).

7.2.2 Execution at Partition Repair (Network Merge)

As a partition of the underlying physical network ceases, which we refer as network merge, the multiple overlays formed during partition, should also be merged i.e. reverse back to

original state. In this section, we assess reversibility of each maintenance strategy and also find the limit (if any), while a network merge happens.

We use a SON of 100 peers. All experiments are done in Mozart-Oz 2.0 [59] in a simulated environment. To simulate the underlying network, the end-to-end delays are set based on the empirical distribution of minimum RTT provided in [49]. The distribution represents significant geographic diversity, so we can say that the simulated SON is geo-distributed. We simulate partition of underlying network in order to create inhospitable operating condition. In the steady state of SON when all nodes are on core ring, we simulate 2, 4 and 10 partitions of the SON, of almost equal sizes. The partition of the SON is withdrawn after 30 seconds and we observe the merging of overlays with time. We have used 3 scenarios, with increasing number of partitions in order to compare and study the reversibility based on number of partitions.

In order to quantify the effect of self-healing, we have used metric: Number of *Islands*. We define an island to be a disconnected sub-graph by following the successor pointer of each node. The number of islands can be \geq number of partitions, due to temporary inaccuracy of the successor pointers or inadequate healing of the maintenance strategy (e.g., as explained in Section 7.2.1, for Figure 7.2b, 3 islands are created, though the number of partitions is 2). The value of this metric should be 1, for a single run, if overlays are merged together after the partition disappears. However, we still continue an experiment until all nodes form a perfect ring, i.e., complete self-healing. Although our representative system, Beernet, allows branches, but increasing branch size affects routing efficiency and increases the probability of creating isolated branches, thus inconsistencies under churn, as explained in [13]. So, for a single run, the ultimate goal is to make “number of islands” as 1 and all nodes to be on core ring. We present the number of islands against time (in second) after the partition disappears and an average of 10 independent runs are taken for each second. It is noticeable that, the termination time of an experiment, apparent in the result, is the maximum among the samples used. A fixed workload is used in an experiment by injecting transactions, modeled as a Homogeneous Poisson Process with $\lambda = 1$ transaction/sec. A transaction reads one key and updates another one.

Sparse Partitioning

In this section we present reversibility results for sparse partitions. Sparse partitions create higher stress on the merge operation than the sequential partitioning. In our simulation, to create P sparse partitions, we create P baskets, where each node of the SON is assigned to a basket with probability $\frac{1}{P}$. The comparative analysis of healing capability of maintenance strategies is portrayed in Figure 7.3 as the average number of islands with time, since the network merge happens. Also, the average number of messages generated for each experiment is shown in Figure 7.4. As we can see in Figure 7.3, the combination of correction-on-* and periodic stabilization is sufficient to achieve reversibility if there is no churn. Merger with passive list, as done in [40], does not show improvement over the combined local maintenance strategies.

7.2. EVALUATION OF MAINTENANCE STRATEGIES

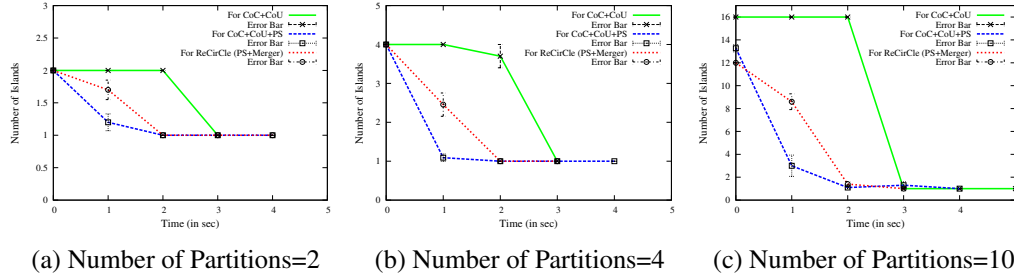


Figure 7.3: Number of islands as a function of time (in sec) starting at the moment of sparse partition repair to assess self-healing using different maintenance strategies

Local Maintenance: Correction-on-*, due to its rapid reaction against an event (join/leave/failure/false suspicion) exhibits certain overlay merging capabilities, however, fails to complete the healing process when the number of partitions goes higher. We can see in Figure 7.3, irrespective of the number of partitions created, correction-on-* has created 1 island once the partition ceases. The reason is that, as a result of partition, each node considers the nodes on other partitions to have failed and adjusts its pointers, however, it continues monitoring those suspected peers. As no churn is experienced, so the list of monitored nodes is not altered. So when it can make a connection with those suspected peers (as partition disappears), a false suspicion event (see Section 2.2) is triggered, as a result of which it corrects its pointers, resulting in merging of overlays. However, as we can see in Figure 7.3c, though the number of underlying partition simulated is 10, there are 16 partitions of the overlay (at $t = 0$), justifying the explanation presented in Section 7.2.1. In terms of overlay merging capability, the correction-on-* has made the number of islands as 1, however it fails to make a perfect ring for most of the runs. An average of 91.5% nodes are on core ring for those runs, where rest of the nodes are on branches. While merging large number of overlays, nodes are placed on branches as an initiation, but as the events (detection of false suspicions in this case) fade away, healing is also discontinued, resulting in branches to remain in the system. However, this is the least-cost healing mechanism as a response of network merge, as shown in Figure 7.4. In comparison with other maintenance principles, the load created on underlying network is much lower for these principles, making it a cheap healing response against any inhospitable event, without creating any inconsistency.

Periodic Stabilization itself is not capable of merging multiple overlays, as also supported by [11]. So, we have combined periodic stabilization with correction-on-*. The period used in our experiments is 1 second. As we can see in Figure 7.3, the combination of reactive and proactive mechanisms show quick response among all. In terms of partition tolerance, we can see the improvement in Figure 7.3c comparing to the correction-on-* mechanisms, however the number of partitions is still more than the induced one, reason for this is explained in Section 7.2.1. The impact of the integration of costly periodic stabilization is clearly visible in Figure 7.4. However, the number of messages decreases with the increase of number of partitions. The reason is that, with the increase of the number of partitions, the size of each independent overlay decreases, resulting in less number of

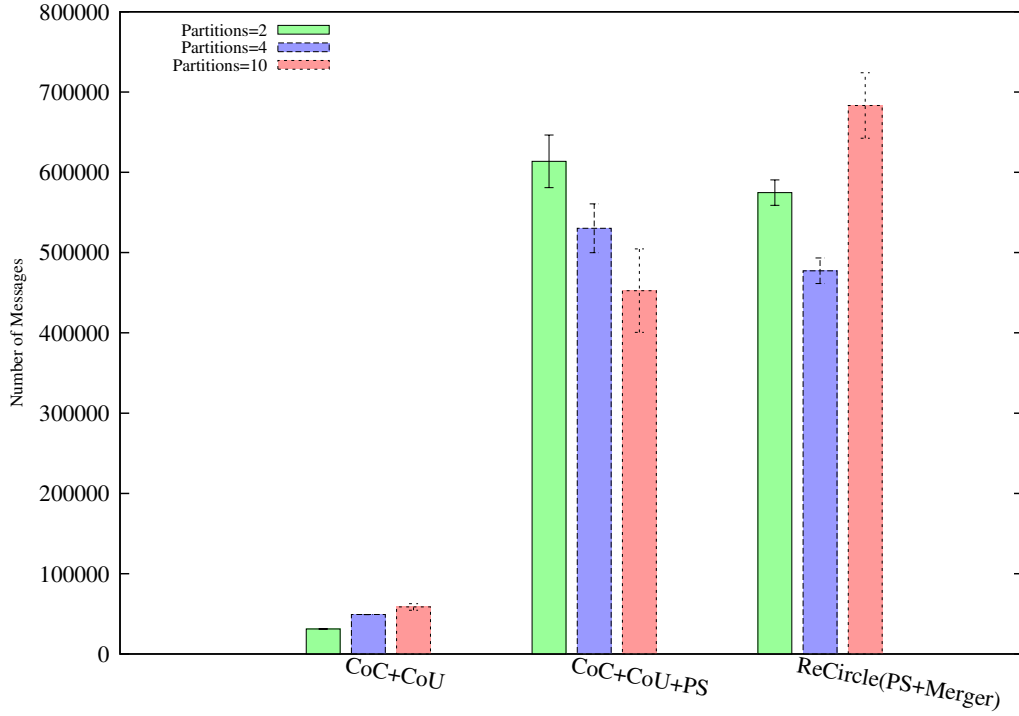


Figure 7.4: Number of Messages generated for 2, 4 and 10 sparse partitions using different maintenance mechanisms

messages to propagate any correction. Thus, any correction within fewer rounds covers an entire small overlay (with fewer nodes) while enduring network partition.

Global Maintenance: As shown in Figure 7.3, the integration of a global maintenance, e.g., merger with passive list, does not show any significant improvement over the combined local healing. The period used to dequeue the elements to generate *mlookups* at each node is 3 seconds and we have kept the *fanout* parameter as 1. However, as we can see in Figure 7.4, this is one of the costliest solutions, even with the *fanout* as 1, especially while merging larger number of overlays. Although, the number of messages is lower for 4 partitions than 2, the reason is due to propagation of corrections by periodic protocol across smaller overlays, as described before, but this effect is overcome by the reactive messages generated in case of 10 partitions. Also, the number of messages generated in case of 2 and 4 partitions are lower than those for the combined local strategies. The reason is the global spreading of corrections by the merger.

Sequential Partitioning

In this section we investigate about overlay merging when a network partitioning, which caused the overlay to split into P contiguous regions (here P is the number of partitions),

7.2. EVALUATION OF MAINTENANCE STRATEGIES

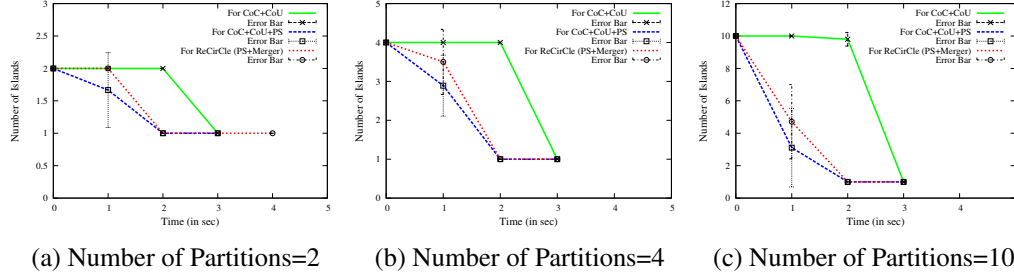


Figure 7.5: Number of islands as a function of time (in sec) starting at the moment of sequential partition repair to assess self-healing using different maintenance strategies

i.e., sequential partitioning, disappears. As already mentioned, merging of a sequential partitioning creates much lesser stress on overlay maintenance than a sparse partitioning. In fact, as a network partition disappears, only 2 nodes in each partition need to modify their pointers, if no churn is experienced during the partition. In our simulation, to create P sequential partitions, we create P baskets, where N/P contiguous nodes on the overlay (by following the successor pointer of each node) are assigned to each basket, here N = total number of nodes on the overlay = 100 in our experiments. Figure 7.5 shows the comparative analysis of reversibility of maintenance strategies. We have used the same metric to assess self-healing: average number of islands with time, since the network merge happens. Also, Figure 7.6 presents the average number of messages generated for each experiment. As we can see in Figure 7.5, all our experiments have achieved complete self-healing irrespective of the number of partitions of the overlay. However, as with sparse partitioning, the combined local maintenance (correction-on-* and periodic stabilization) shows quick response among all. The integration of a reactive global maintenance, e.g., merger with passive list, as done in [40], does not show any improvement over the combined local maintenance strategies.

Local Maintenance: Unlike sparse partitioning, while repairing sequential partitions, the reactive local maintenance, correction-on-* principles are able to achieve reversibility even for higher level of partitioning, as evident in Figure 7.5. The reason is the same as described for sparse partitioning, the rapid reaction of these principles against any event (join/leave/failure/false suspicion). As we can notice in Figure 7.5, the number of partitions in the system at $t = 0$, is the same as the number of simulated partitions of the underlying network. However, correction-on-* principles fail to provide partition-tolerance for any of these scenarios, as unavailability of key ranges are introduced, also the nodes in each partition fail to form a ring overlay in each partition, as described in Section 7.2.1. In terms of overlay merging capability, the correction-on-* principles are sufficient for sequential partitioning of any number, even offer the least-cost complete self-healing among all, as portrayed in Figure 7.6. Though these principles show slowest response among all, but the results obtained is consistent irrespective of the number of partitions in the system, i.e., for all the runs, the healing is completed within 3 time unit, since the network partition ceases.

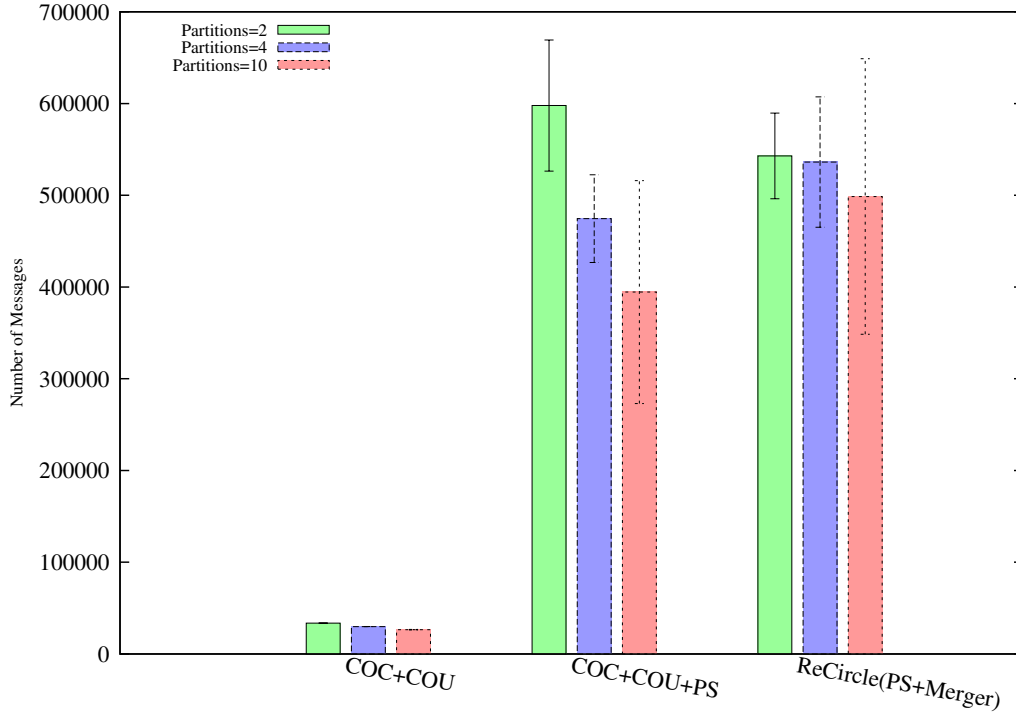


Figure 7.6: Number of Messages generated for 2, 4 and 10 sequential partitions using different maintenance mechanisms

Figure 7.5 shows, as with sparse partitioning, the combination of correction-on-* and periodic stabilization continues the trend of showing the quickest response among all while merging sequential partitions. The period used in our experiments is 1 second. This combined local maintenance provides partition-tolerance for all the runs of all levels of partitioning. In terms of reversibility, this combination provides the quickest response and convergence among all. However, this is one of the costliest self-healing, as evident in Figure 7.6, making it explicit the bandwidth consumption of the costly periodic stabilization.

Global Maintenance: As Figure 7.5 shows, the integration of a global maintenance, e.g., merger with passive list, with periodic stabilization does not show any improvement over the combined local healing. On the contrary, at times the convergence time (see Figure 7.5a) and bandwidth consumption (see Figure 7.6 for 4 and 10 partitions) are more than the combined local healing. The period used to dequeue the elements to generate *mlookups* at each node is 3 seconds and we have kept the *fanout* parameter as 1.

7.3 Related Work

Several versions (through gradual improvement) of overlay merge algorithm are proposed in [40], of which ReCircle (described in Section 3.2.3) is adapted in Beernet++ for the pur-

7.4. DISCUSSION

pose of this work. The evaluation, carried out in these works, is to validate the proposed algorithms and sensitivity analysis of different parameters of these algorithms on convergence time and bandwidth consumption. The objective of our work is to assess self-healing achievable using different maintenance strategies and also to identify the properties/limit of the maintenance operation to survive in an inhospitable environment caused by high level of network partition. So, the work in [40] and our work complement each other.

In [81], a centralized approach using bootstrap server is proposed to detect multiple overlays and initiate merge, as the underlying network partition ceases. As per this approach, peer with the smallest virtual identifier, periodically sends message to a bootstrap server. As the bootstrap server receives multiple messages, it detects multiple overlays in the system, which then informs all peers to initiate merging process. This approach depends on a central bootstrap server. Also, this work lacks a full algorithm and evaluation of merge process. The protocols for network partitions and merge are proposed in [83], at the core of which is a broadcast protocol. A node sends broadcast message to d uniformly selected nodes to gather knowledge about the network it belongs to, each node, which receive such broadcast message repeats the process. However, this work does not specify how a node acquires knowledge about the broadcast candidates and satisfy uniform sampling. Also, the merge protocol presented states that nodes of one overlay join the other, does not mention the process to detect a network merge and how to decide which overlay to trigger the join process. Merging of multiple P-Grid [84] SONS is presented in [85], [86], which is complementary to our work. Methods/algorithms to merge two independently bootstrapped peer-to-peer overlays are presented in several works [87], [88].

7.4 Discussion

In this chapter, we have investigated about network partitioning. We have considered scenarios, where no churn is experienced by the overlay during a network partition, which correspond to the network partitions of short durations. We have presented comparative analysis of partition-tolerance and reversibility of existing maintenance strategies, namely, correction-on-*, periodic stabilization and ReCircle (as done in [40]). We have observed that a partition can occur even if there is no communication problem; also have identified and verified the pre-conditions to ensure partition tolerance for any scenario of network partition. Our results show that micro-level interactions among nodes (i.e., local corrections at each node) are able to trigger and accomplish macroscopic healing (i.e., merging of multiple overlays). However, the local corrections need to be both proactive and reactive to attain better recovery, especially when there are high levels of sparse partitioning of the overlay. The result obtained only through such local corrections, without any explicit merge algorithm, is competitive (or at times better) with the one with an explicit overlay merger.

Chapter 8

Interaction between Network Partitioning and Churn

In Chapter 7, we have considered network partition and merge, where there was no churn in between, which is not a realistic scenario, since for a peer-to-peer network churn events are the most usual ones. Though in most existing applications churn remains under a certain limit, as per studies [29, 30, 31], systems with low/average churn face high peaks and this may happen even during short duration of a network partition. Consider the scenario of a *Structured Overlay Network (SON)* running on mobile phones or on an ad hoc network. In such a dynamically changing environment network partition can be a frequent event, along with high churn. However, we have not found any work in literature that demonstrates reversibility for such inhospitable environment, where a SON goes through a network partition, while facing churn at the same time.

In this chapter, We propose a model, namely “Stranger Model”, to generalize the impact of *simultaneous* network partition and churn. We show that this interaction causes partitions to eventually become strangers to each other, which makes full reversibility impossible when this happens. Using this model, we can predict when irreversibility arrives, which we verify via simulation. Later, we evaluate the reversibility of maintenance principles while facing churn during a network partition and identify the preconditions to achieve reversibility. In this chapter, we have used only sparse partitioning for our experiments, as sparse partitions create higher stress on the maintenance than the sequential partitioning.

8.1 Stranger Model

Before the network partition and after network merge, churn is handled by any SON as usual. It is churn during a network partition, which creates challenge for merging the overlays as network merge happens. We propose a model, namely “Stranger Model”, to understand the impact of churn during a network partition. Using this model we quantify the challenge for the maintenance mechanism, while merging multiple overlays (created while enduring the network partition), as the partition ceases.

We use the same definition of churn as in Section 3.1: percentage of nodes turnover

per time unit (seconds). If we assume equal probability of join/leave event and a single event per time unit, then every other time unit, a node will leave and a new node will join the network, i.e., every other time unit the total number of peers will be the same, whereas only a single node has a changed identity. So, during network partition, every other time unit, a new node replaces an existing node in a partition, about which no other node of any other partition has any knowledge. We can say that this new node becomes a stranger for the nodes on other partition. In our model, we assume uniform distribution of lifetime of peers. However, in real systems, this is necessarily not the case. So, our model is a pessimistic one, in other terms corresponds to the “worst case” scenarios. Investigation using a more realistic up-time distribution of peers is left as future work.

Suppose, the number of nodes on a SON is N . For simplicity we will consider only 2 partitions of almost equal size, to present our model, can be generalized for N partitions as well. After a network partition, two independent overlays, P_1 and P_2 are formed having n_1 and n_2 nodes respectively on each overlay, i.e., $n_1 + n_2 = N$, $n_1 \approx n_2$ and P_i is the set of nodes of partition i . The best possible starting state for a partition of the overlay is, when each partition has complete knowledge about the other, i.e., $(\bigcup_{p_i \in P_1} KB_{p_i}) \cap P_2 = P_2$. We present our model to generalize the interaction between network partition and churn, during the partition period, based on this simplifying assumption. For a churn intensity of $C\%$, after 1 time unit, the number of nodes on P_2 known to P_1 is, $(\bigcup_{p_i \in P_1} KB_{p_i}) \cap P_2 = n_2 \times e^{-\frac{C}{100}}$. After t time units, $(\bigcup_{p_i \in P_1} KB_{p_i}) \cap P_2 = n_2 \times e^{-\frac{Ct}{100}}$.

Prediction of Irreversibility: Using stranger model, we can predict the limit (in time unit) of achieving reversibility, i.e., the number of time units, since a network partition, beyond which the system is unable to achieve reversibility by itself. For every principle, healing by merging of overlays is based on the known references of peers on other partition, with which communication can be established as the partition ceases. So, with the increment of strangers on both P_1 and P_2 , the merging becomes more difficult and at time unit T_{CO} , $(\bigcup_{p_i \in P_1} KB_{p_i}) \cap P_2 = \emptyset$ (and vice versa). After T_{CO} time unit P_1 and P_2 will be complete strangers to each other and as the nodes on a partition have no reference about any peer on the other partition, no healing mechanism of the system will be effective. We will refer T_{CO} as the *cut-off* point, beyond which, mechanism outside the system or third party intervention is required to make the system reversible. We can derive T_{CO} as a function of C and n_2 using our model:

$$n_2 \times e^{-\frac{CT_{CO}}{100}} = 1 \therefore T_{CO} = \frac{100 \times \ln n_2}{C} \quad (8.1)$$

Using Equation 8.1, we can derive T_{CO} for P_2 , with respect to P_1 for a given C . We present experimental validation of our model for churn of 10%, 30% and 80%. For these experiments, we use a SON of 100 peers. To simulate the underlying network, the end-to-end delays are set based on the empirical distribution of minimum RTT provided in [49]. During steady state, a network partition is simulated by creating 2 partitions of the overlay of equal size, i.e., $|P_1| = |P_2| = 50$. We have verified that after the partition, i.e., at $t = 0$, $(\bigcup_{p_i \in P_1} KB_{p_i}) \cap P_2 = P_2$ and vice versa, for these experiments. Churn of particular intensity is injected for t seconds. To inject a churn event, a partition is chosen

8.1. STRANGER MODEL

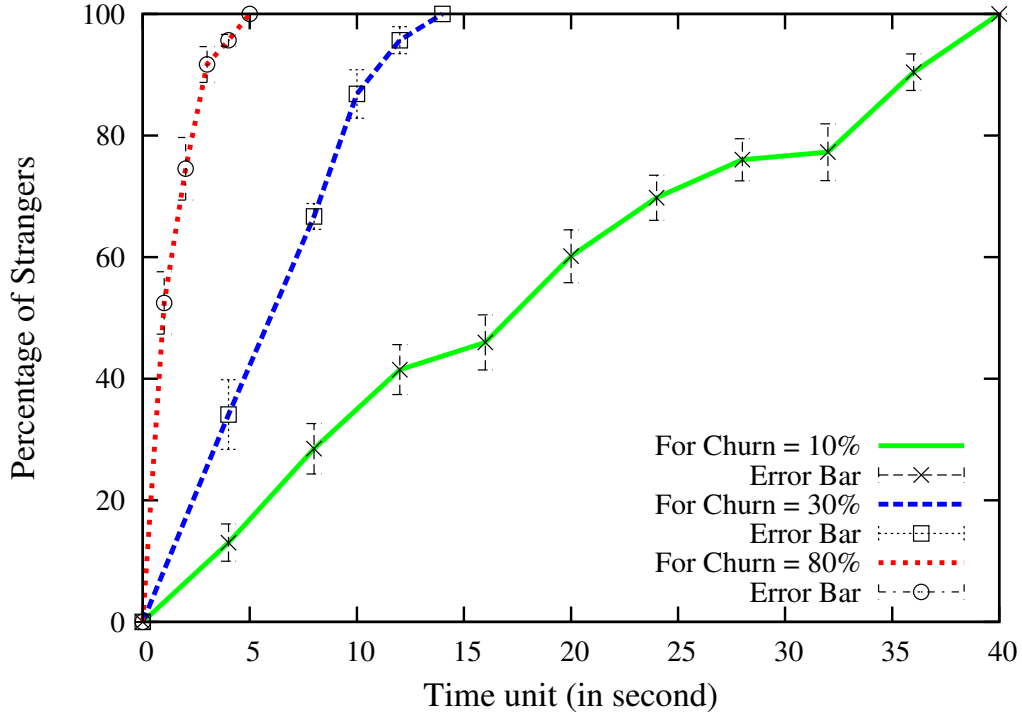


Figure 8.1: Evaluation of Stranger model for 10%, 30% and 80% of churn

Churn (C)	Theoretical <i>Cut-off</i> Time	Measured <i>Cut-off</i> Time
10%	39.12	40
30%	13.04	14
80%	4.89	5

Table 8.1: *Cut-off* time (T_{CO}) for different values of Churn

with equal probability. We have used correction-on-change, periodic stabilization and the merger with passive KB for maintenance to have the best measurement of strangers. After withdrawing churn, we retrieve knowledge base of each node of P_1 and make a superset of those. We count the nodes in P_2 which are not in that set and calculate the percentage, this is the percentage of nodes in P_2 , which are stranger to P_1 . We do the same for P_1 with respect to P_2 and report the average for a single run. An average of 10 such runs is reported in Figure 8.1 with increasing time. We present the values of T_{CO} for 10%, 30% and 80% of churn and $n_1 = n_2 = 50$ using Equation 8.1 and from experiments in Table 8.1. As shown in Figure 8.1, the percentage of strangers increases with time, for all values of churn. Also, the *Cut-off* points coincide with those derived using Equation 8.1, as presented in Table 8.1, thus validating our model. We have expressed T_{CO} assuming the best possible starting state for a partition of the overlay, i.e., each partition has converged knowledge about the other. A generic expression for T_{CO} is subject to future work.

CHAPTER 8. INTERACTION BETWEEN NETWORK PARTITIONING AND CHURN

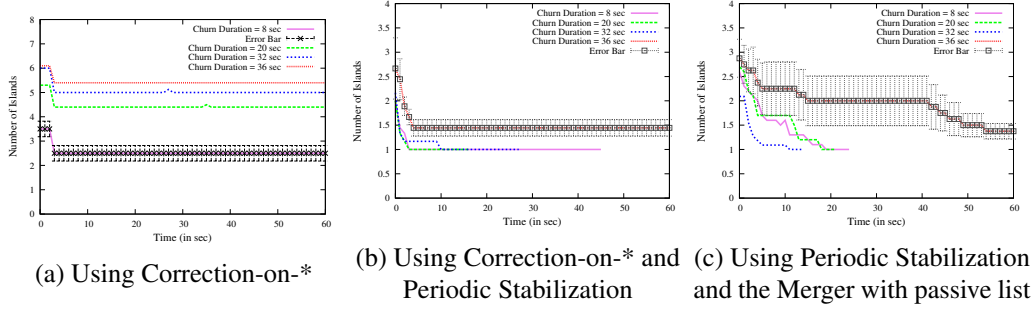


Figure 8.2: Number of islands as a function of time (in sec) after withdrawing churn and partition to assess self-healing against strangers using different maintenance strategies

8.2 Evaluation of Maintenance Principles

We evaluate maintenance strategies in terms of their abilities to overcome the challenges posed by strangers, while merging multiple overlays. The generalized effect of churn of different intensities during network partition is the rate of increasing the number of strangers. So, in these experiments, we have used only one value of churn, namely 10% of churn for different durations to create desired percentages of strangers in the system. We use similar experimental setup as described in Section 8.1. We have continued the injection of 10% churn for 8, 20, 32 and 36 seconds, since the network partition is introduced. Then we withdraw churn, wait for 30 seconds for the healing of the effect of churn on both partitions and restore the network partition. We observe the reversibility of different maintenance mechanisms with time, using the same metric as in Section 7.2.2: number of islands. We present the average of 10 sample runs in Figure 8.2; however, we have excluded samples (especially for experiments with 36 seconds) for which the partitions become complete strangers (i.e., $(\bigcup_{p_i \in P_1} KB_{p_i}) \cap P_2 = \emptyset$ and vice versa). For stranger measurements, we have constructed a passive KB at each node for these experiments, but the KB is not used for any correction/healing (i.e., to trigger the merger of ReCircle).

8.2.1 Correction-on-*

We can see in Figure 8.2a, correction-on-* principles fail to merge the overlays even with the lowest number of strangers in the system. Also, as the duration of churn increases i.e. increment of strangers, so is the number of partitions, which remains the same throughout an experiment, after the initial healing during first 3 seconds. This is due to the lack of liveness property of these maintenance mechanisms that self-healing is discontinued.

8.2.2 Correction-on-* and Periodic Stabilization

After integration of periodic stabilization, we can see significant improvement, as apparent in Figure 8.2b. The correction-on-* mechanisms, along with periodic stabilization shows reversibility, even for churn duration of 32 seconds. However, beyond that, this combined

8.2. EVALUATION OF MAINTENANCE PRINCIPLES

strategy fail to guarantee reversibility, i.e., for 36 seconds of churn duration, there are runs, for which merging has not converged. We have observed till 120 seconds for these runs, but the result remains the same throughout, so for presentation purpose till 60 seconds is shown.

8.2.3 ReCircle (Periodic Stabilization and Merger with passive list)

As shown in Figure 8.2c, ReCircle gives no more improvement over combined local maintenance (Figure 8.2b). It shows reversibility up to churn duration of 32 sec. However, for the experiment with churn duration of 36 sec, there are runs, which have not converged. We have observed till 120 sec, the result remains the same, so for presentation purpose, till 60 sec is shown. The reason is the lack of knowledge to trigger the merger using the passive list.

8.2.4 Knowledge Base

ReCircle (Periodic Stabilization and Merger with passive list) works well for up to 32 sec churn, i.e., 77% strangers (see Figure 8.1), whereas adding a passive Knowledge Base works up to 36 sec churn, i.e., 90% strangers (and beyond, verification of which is subject to future work), as shown in Figure 8.3a. This is much closer to the limit of 100% strangers that is reached at 40 sec churn. This clearly shows the effectiveness of the Knowledge Base. We have used $\sigma = 1$ second to optimistically use elements of the knowledge base to trigger the merger. Also, we have integrated the correction-on-change principle (Section 3.2.1) to avoid any inconsistency while handling churn using periodic stabilization as the only local correction policy (as discussed in Section 3.2.2), without causing any extra load on bandwidth consumption. Using all three mechanisms (Correction-on-*, ReCircle, and Knowledge Base) gives fast convergence of number of islands to 1. Using only two of these three mechanisms will either not converge to 1 or else converge much slower to 1. We have excluded samples for Figure 8.3a, for which the partitions become complete strangers (i.e., $(\bigcup_{p_i \in P_1} KB_{p_i}) \cap P_2 = \emptyset$ and vice versa). Because for these scenarios, this combined healing falls short, as there is no knowledge in a partition about the other to achieve reversibility.

8.2.5 Oracle

As we can see in Figure 8.3b, when the partitions become complete stranger to each other, it is possible to achieve reversibility, as long as an oracle injects the lost knowledge about the other partition. We have implemented an oracle in the application layer, which every 5-second, picks up 2 pairs of nodes from its list and introduce them to each other through an API. The list of peers at application layer can be built by either active or passive way. As per the active approach, the application layer can periodically retrieve knowledge base from the peers it knows, thus build a superset of these knowledge bases, whereas in passive approach the application layer comes to know about peers while joining. In our experiments, we have used the second approach to build the list at application layer. Figure 8.3b shows the convergence for ≥ 40 seconds churn, i.e., 100% strangers (see Figure 8.1). Also, for

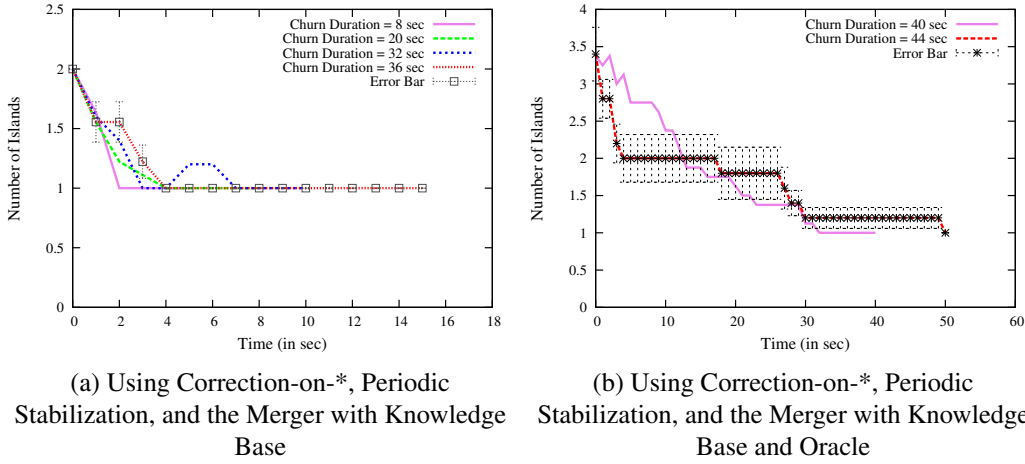


Figure 8.3: Number of islands as a function of time (in sec) after withdrawing churn and partition to assess self-healing against strangers using different maintenance strategies

these experiments, we have restored network partition immediately, instead of waiting for 30 seconds, in order to observe the overall impact the oracle has on the healing process. For this reason, in the first couple of snapshots the number of islands is higher than 2, as a result of the temporary inaccuracy in the neighborhood of each node, due to churn.

8.3 Recovery Time and Cost

We present recovery time and cost, in term of number of messages, to achieve reversibility against increasing strangers among the partitions of the system. We have used similar experimental setup and induced churn of 10% for increasing duration during the partition. Correction-on-* and ReCircle with the knowledge base approach (passive KB and oracle) are used as part of maintenance for these experiments. After withdrawing churn, partition is restored and we measure the time (in seconds) required for complete healing, i.e., all nodes organized into a perfect ring topology. We have used mean value of 20 independent runs for every 4 seconds increase of churn duration. Figure 8.4 shows the result. This along with Figure 8.1 gives an idea regarding the percentage of strangers in the system and the time required to achieve reversibility. As expected, the recovery time increases with the number of strangers in the system. We report the average number of messages generated for increasing churn duration in Figure 8.5. This also follows the similar pattern as in Figure 8.4. We notice large number of messages generated for a network of 100 peers, raising concern about scalability of the system. However, by controlling the number of messages triggered by the merger of ReCircle using the knowledge base and oracle parameters and setting an optimistic period for periodic stabilization, the bandwidth consumption can be lowered, while trading-off convergence time. Also, it is essential that the system should adapt with the operating environment, which we keep as future work.

8.4. RELATED WORK

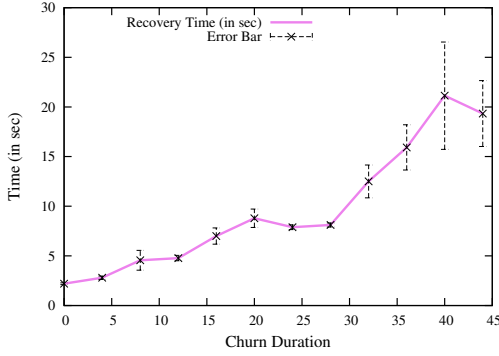


Figure 8.4: Recovery/Healing time for increasing strangers

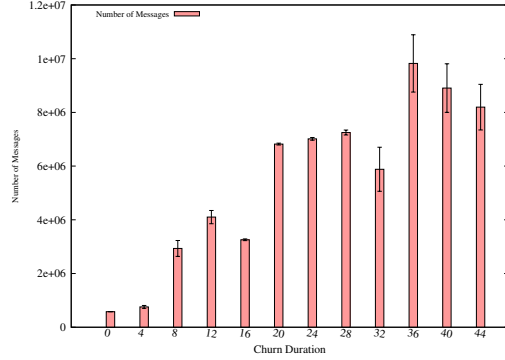


Figure 8.5: Number of messages generated for increasing strangers

8.4 Related Work

Several works are done to assess resilience of various maintenance strategies under churn [89], [63], [90], [58]. However, we have not found any work that analyzes the impact of the interaction of network partition and churn, thus to quantify the challenges posed by this interaction on the maintenance strategy of the overlay. In our work, we have presented a model, using which it is possible to generalize the effect of network partition under churn of any intensity and duration. Also, this can provide useful information to the application layer regarding the cut-off point, beyond which the application or third-party intervention is required to take initiative by injecting knowledge, in order to achieve reversibility.

8.5 Discussion

In this chapter, we have investigated the interaction between *Network Partitioning* and *Churn (node turnover)* in *Structured Overlay Networks*. We have proposed a model, namely “Stranger Model”, to generalize the impact of *simultaneous* network partition and churn. We have shown that this interaction causes partitions to eventually become strangers to each other, which makes full reversibility impossible when this happens. Using stranger model, we have also identified the boundary (cut-off point), beyond which the system is unable to achieve reversibility by itself. We propose to use a knowledge base to handle partitions under churn, that contains knowledge collected in a passive way at each node and injected by an oracle beyond the cut-off point.

Chapter 9

Conclusions and Future Work

In this chapter, we summarize our contributions. We conclude by discussing our future works, in continuation towards our objective. Also, we mention some research ideas derived from the results obtained so far.

9.1 Reversible and Predictable System

The advent and demand for new technologies consistently increasing the complexity of distributed systems. The heterogeneity and inclusion of increasing mobility in these systems continues to make the operating environments more inhospitable. In order to build reliable complex systems, it is imperative to assess and identify required properties of the maintenance strategy of such systems to achieve complete self-healing in the entire operating space. In this thesis, we have introduced the concept of *Reversibility*: functionality of a system is a property of current operating environment, and is not affected by the failures in the past. Reversibility allows opening new venues for application design, such as mobile and ad hoc networks and Internet of Things, for which existing fault-tolerance techniques are insufficient. For the purpose of this work, we have chosen one particular class of complex systems, namely a class of *Structured Overlay Networks (SONs)*, defined by the reference architecture of [18]. To our knowledge, existing literature lacks an assessment or verification of SONs reversibility by ensuring complete healing of the predefined structure. We have organized the entire operating space of our representative complex systems by identifying the stress parameters. We have introduced a new maintenance principle, namely *Knowledge Base*, which is required to survive and achieve reversibility against inhospitable environments.

In this thesis, we have investigated about two stress parameters: Churn and Network Partitioning, also interaction between them. As a part of this journey, we have done comparative analysis of the healing capability of existing maintenance principles of ring-based SONs, while facing inhospitable environments caused by these two stress parameters. We have chosen Beernet [13] as a representative example of a large class of ring-based SONs [18] and analyzed its healing capability by enabling different combinations of principles. We have identified the preconditions to achieve reversibility for ring-based SON. Also, the

demonstration of a reversible SON, *Beernet++*, against churn and network partitioning is presented in this thesis. We have observed that a partition can occur even if there is no communication problem; also have identified and verified the preconditions to ensure partition tolerance for any scenario of network partition. Our results show that micro-level interactions among nodes (i.e., local corrections at each node) are able to trigger and accomplish macroscopic healing (i.e., merging of multiple overlays). However, the local corrections need to be both proactive and reactive to attain better recovery, especially after sustaining simultaneous network partition and churn. The result obtained only through such local corrections, without any explicit merge algorithm, is competitive with the one with an explicit overlay merger.

Using the “Stranger model”, we have generalized the effect of the interaction between network partition and churn. The stranger model also allows *predicting* when irreversibility arrives (the *cut-off* point) due to *simultaneous* network partition and churn. This can provide useful information for applications, thus support building application in very hostile environments. We propose to use knowledge base principle to handle partitions under churn, that contains knowledge collected in a passive way at each node and injected by an oracle beyond the cut-off point.

On our way towards the objective of this thesis, we have observed and analyzed other macro-level phenomenon, for example, phase transition in the system. We have shown that *Beernet++* does reversible phase transitions, i.e., it “boils” to the gaseous state (becomes disconnected) when churn increases and condenses from gaseous back to solid phase as churn intensity goes down. We also identify and analyze the apparent “critical points” from the experiments while doing such transitions. The concept of phase, phase transition and critical point allows designing applications to work in inhospitable environments. Also, as a prerequisite of this work, a QoS-aware self-adaptable eventually perfect failure detection algorithm is presented and evaluated in this thesis. Finally, we present the evaluation of several high-level properties of *Beernet++* for inhospitable environments.

9.2 Future Work

In this section, we discuss about some ideas towards our overall objective: “Design *Predictable* and *Reversible* systems”. Some of these ideas are continuation of the works presented in this thesis, others explore the dimensions of the operating space of a complex system, not addressed in this thesis.

9.2.1 API and Phase Transitions

In this thesis, we have presented our first investigation about phase and phase transitions in a SON with churn. We intend to investigate further the analogy between phase in SONs and in physical systems. We are in the process of designing an API that makes explicit the concept of phase and phase transitions. The API will expose the maximum functionality and information to the application layer, widening the way to build applications that run in inhospitable environments. In future work, we will investigate applications that take advantage of this API to survive in extremely hostile environments.

9.2. FUTURE WORK

9.2.2 Maintenance Principles and Efficient Self-Healing

In future work, We intend to improve the maintenance principles, e.g., we intend to use gossip protocols to improve the knowledge base technique. Also, in our experiments, we have introduced some parameters, e.g., *Join Timeout* for the new peers to issue repeated join requests, we intend to improve these parameters, which are currently empirical. Besides, it is necessary for efficient survival/self- healing in an inhospitable environment to adapt such system parameters as per the operating environment. This also requires a study to identify the influences of different parameters on convergence time of healing or phase transitions/critical points.

9.2.3 Network Dynamicity and its Impact

In this thesis, we have not investigated about how a highly dynamic underlying network affects the maintenance of a system. Also, we intend to study the interaction of this dimension of operating space with other stress parameters (e.g., churn, network partition).

9.2.4 Experimentation and Validation on Real-World Environment

In future work, we intend to evaluate our system in real-world (non-simulated) dynamic environment, like PlanetLab [91] and Community Networks, to verify the findings from simulation.

Bibliography

- [1] Richard John Anthony. Emergence: A paradigm for robust and scalable distributed applications. In *In Proceedings of IEEE International Conference on Autonomic Computing (ICAC'04)*, pages 132–139, 2004.
- [2] T. De Wolf, G. Samaey, T. Holvoet, and D. Roose. Decentralised autonomic computing: Analysing self-organising emergent behaviour using advanced numerical methods. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 52–63, June 2005.
- [3] O. Babaglu, M. Jelasity, T. Holvoet, and D. Roose. *Unconventional Programming Paradigms*, volume 3566, chapter Grass- roots Approach to Self-management in Large-Scale Distributed Systems, pages 286–296. Springer Berlin/Heidelberg, August 2005.
- [4] Napster. Inc. napster. <http://www.napster.com>, 1999. Accessed: 2015.
- [5] Audiogalaxy. The Audiogalaxy satellite. <http://www.audiogalaxy.com/>, 2001. Accessed: 2004.
- [6] OpenNap Community. Open source napster server. <http://opennap.sourceforge.net/>, 2001. Accessed: 2015.
- [7] Gnutella. Gnutella. <http://www.gnutella.com/>, 2006.
- [8] FreeNet Community. The freenet project. <http://freenetproject.org>, 2003. Accessed: 2015.
- [9] Krishna Gummadi, Ramakrishna Gummadi, Steve Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The impact of dht routing geometry on resilience and proximity. pages 381–394. ACM Press, 2003.
- [10] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. pages 71–76, 2001.
- [11] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*, pages 149–160, 2001.

BIBLIOGRAPHY

- [12] Luc Onana Alima, Sameh El-ansary, Per Brand, and Seif Haridi. Dks(n,k,f): a family of low communication, scalable and fault-tolerant infrastructures for p2p applications. In *IN 3RD IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGRID)*, pages 344–350, 2003.
- [13] Boris Mejías Candia. *Beernet: A Relaxed Approach to the Design of Scalable Systems with Self-Managing Behaviour and Transactional Robust Storage*. PhD thesis, ICTEAM, Université catholique de Louvain, Louvain-la-Neuve, Belgium, October 2010.
- [14] B. Mejías and P. Van Roy. Beernet: Building self-managing decentralized systems with replicated transactional storage. *IJARAS: International Journal of Adaptive, Resilient and Automatic Systems*, July 2010.
- [15] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.
- [16] Marco Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1):45–67, March 1993.
- [17] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.
- [18] Karl Aberer, Luc Onana Alima, Ali Ghodsi, Sarunas Girdzijauskas, Seif Haridi, and Manfred Hauswirth. The essence of p2p: A reference architecture for overlay networks. In *IN P2P2005, THE 5TH IEEE INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING*, pages 11–20, 2005.
- [19] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. A structured overlay for multi-dimensional range queries. In *Proceedings of the 13th International Euro-Par Conference on Parallel Processing*, Euro-Par’07, pages 503–513, Berlin, Heidelberg, 2007. Springer-Verlag.
- [20] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS’03, pages 9–9, Berkeley, CA, USA, 2003. USENIX Association.
- [21] M.Frans Kaashoek and DavidR. Karger. Koorde: A simple degree-optimal distributed hash table. In M.Frans Kaashoek and Ion Stoica, editors, *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 98–107. Springer Berlin Heidelberg, 2003.
- [22] Valentin Mesaros, Bruno Carton, and Peter Van Roy. P2ps: Peer-to-peer development platform for mozart. In Peter Van Roy, editor, *Multiparadigm Programming in Mozart/Oz*, volume 3389 of *Lecture Notes in Computer Science*, pages 125–136. Springer Berlin Heidelberg, 2005.

BIBLIOGRAPHY

- [23] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4):353–366, August 2004.
- [24] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 353–366, New York, NY, USA, 2004. ACM.
- [25] B. Leong, B. Liskov, and E. Demaine. EpiChord: Parallelizing the chord lookup algorithm with reactive routing state managements. In *Proceedings of the ACM SIGCOMM 2004 Symposium on Communication, Architecture, and Protocols*, Singapore, November 2004. IEEE Computer Society.
- [26] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. Bandwidth-efficient management of dht routing tables. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 99–114, Berkeley, CA, USA, 2005. USENIX Association.
- [27] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, Seattle, WA, USA, March 2003. USENIX.
- [28] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 163–170, New York, NY, USA, 2000. ACM.
- [29] Ranjita Bhagwan, Stefan Savage, and GeoffreyM. Voelker. Understanding availability. In M.Frans Kaashoek and Ion Stoica, editors, *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 256–267. Springer Berlin Heidelberg, 2003.
- [30] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, January 2002.
- [31] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 189–202, New York, NY, USA, 2006. ACM.
- [32] M. Steiner, T. En-Najjary, and E.W. Biersack. Long term study of peer behavior in the kad dht. *Networking, IEEE/ACM Transactions on*, 17(5):1371–1384, Oct 2009.
- [33] Farnam Jahanian, Craig Labovitz, and Abha Ahuja. Experimental study of internet stability and wide-area backbone failures. Technical Report CSE-TR-382-98, University of Michigan, November 1998.

BIBLIOGRAPHY

- [34] David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do internet services fail, and what can be done about it? In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.
- [35] Vern Paxson. End-to-end routing behavior in the internet. *SIGCOMM Comput. Commun. Rev.*, 36(5):41–56, October 2006.
- [36] Taiwan quake exposes internet vulnerability. <http://www.globalsecuritynews.com/Asia/Wolfe-Adam/Taiwan-Quake-Exposes-Internet-Vulnerability>, January 2007. Accessed: 2015.
- [37] ISP quarrel partitions Internet. <http://www.wired.com/threatlevel/2008/03/isp-quarrel-par/>, March 2008. Accessed: 2015.
- [38] The Cogent-Level 3 Dispute. <http://www.lookingglassnews.org/viewstory.php?storyid=2883>, October 2005. Accessed: 2015.
- [39] Karl Aberer, Anwitaman Datta, and Manfred Hauswirth. Route maintenance overheads in DHT overlays. In *Workshop on Distributed Data and Structures*, 2003.
- [40] Tallat M. Shafat. *Partition Tolerance and Data Consistency in Structured Overlay Networks*. PhD thesis, KTH Royal Institute of Technology, Sweden, 2013.
- [41] Ruma R. Paul, Peter Van Roy, and Vladimir Vlassov. An empirical study of the global behavior of a structured overlay network. In *Peer-to-Peer Computing (P2P), 14-th IEEE International Conference on*, pages 1–5, Sept 2014.
- [42] Michael J. Freedman, Karthik Lakshminarayanan, Sean Rhea, and Ion Stoica. Non-transitive connectivity and DHTs. In *Proceedings of the 2nd Workshop on Real Large Distributed Systems*, 2005.
- [43] Ali Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD thesis, KTH Royal Institute of Technology, Sweden, 2006.
- [44] Supriya Krishnamurthy and John Ardelius. An analytical framework for the performance evaluation of proximity-aware structured overlays. Technical report, Swedish Institute of Computer Science (SICS), 2008.
- [45] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-man: Gossip-based fast overlay topology construction. *Comput. Netw.*, 53(13):2321–2339, August 2009.
- [46] Programming Languages and Distributed Computing Research Group. Beernet: pbeer-to-pbeer network. <http://beernet.info.ucl.ac.be>, 2009. Accessed: 2015.
- [47] Christian Cachin, Rachid Guerraoui, and Luis Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer, second edition, 2011.

BIBLIOGRAPHY

- [48] Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera. On the quality of service of failure detectors. *IEEE Trans. Comput.*, 51(5):561–580, May 2002.
- [49] Jay Aikat, Jasleen Kaur, F. Donelson Smith, and Kevin Jeffay. Variability in TCP round-trip times. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, pages 279–284, New York, NY, USA, 2003. ACM.
- [50] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 354–363, 2002.
- [51] Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, and Theo Ungerer. A new adaptive accrual failure detector for dependable distributed systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 551–555, New York, NY, USA, 2007. ACM.
- [52] Network Working Group. RFC 2988 : Computing TCP's retransmission. <http://www.rfc-editor.org/rfc/rfc2988.txt>, 2000.
- [53] Mikel Larrea, Sergio Arévalo, and Antonio Fernández. Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 34–48, London, UK, UK, 1999. Springer-Verlag.
- [54] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *Dependable Computing, 2001. Proceedings. 2001 Pacific Rim International Symposium on*, pages 146–153, 2001.
- [55] C. Fetzer, U. Schmid, and M. Susskraut. On the possibility of consensus in asynchronous systems with finite average response times. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 271–280, June 2005.
- [56] Tallat M. Shafaat, Monika Moser, Thorsten Schütt, Alexander Reinefeld, Ali Ghodsi, and Seif Haridi. Key-based consistency and availability in structured overlay networks. In *Proceedings of the 3rd International Conference on Scalable Information Systems*, InfoScale '08, pages 13:1–13:5, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [57] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell, and Seif Haridi. A statistical theory of chord under churn. In *Proceedings of the 4th International Conference on Peer-to-Peer Systems*, IPTPS'05, pages 93–103, Berlin, Heidelberg, 2005. Springer-Verlag.
- [58] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell, and Seif Haridi. An analytical study of a structured overlay in the presence of dynamic membership. *IEEE/ACM Transactions on Networking (TON)*, 16(4):814–825, August 2008.

BIBLIOGRAPHY

- [59] Mozart Consortium. The Mozart-Oz programming system. <http://mozart.github.io/>, 2013. Accessed:2015.
- [60] Miguel Castro, Manuel Costa, and Antony Rowstron. Performance and dependability of structured peer-to-peer overlays. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, DSN '04, pages 9–, Washington, DC, USA, 2004. IEEE Computer Society.
- [61] Antony Rowstron and Peter Druschel. Pastry:scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.
- [62] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell, and Seif Haridi. Comparing maintenance strategies for overlays. In *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, PDP '08, pages 473–482, Washington, DC, USA, 2008. IEEE Computer Society.
- [63] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*, PODC '02, pages 233–242, New York, NY, USA, 2002. ACM.
- [64] S. El-Ansary, E. Aurell, P. Brand, and S. Haridi. Experience with a physics-style approach for the study of self properties in structured overlay networks. In *Proc. SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, May 2004.
- [65] Sameh El-Ansary, Erik Aurell, and Seif Haridi. A physics-inspired performance evaluation of a structured peer-to-peer overlay network. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks, part of the 23rd Multi-Conference on Applied Informatics, Innsbruck, Austria, February 15-17, 2005*, pages 116–122, 2005.
- [66] S. Apel and K. Böhm. Self-organization in overlay networks. In *Proceedings of CAISE Workshop Adaptive and Self-Managing Enterprise Applications*, volume 2, pages 139–153, 2005.
- [67] Joseph S. Kong, Jesse S.A. Bridgewater, and Vwani P. Roychowdhury. Resilience of structured p2p systems under churn: The reachable component method. *Computer Communications*, 31(10):2109–2123, June 2008.
- [68] Wikipedia. Phase (matter). https://en.wikipedia.org/wiki/Phase_%28matter%29, July 2015.
- [69] Peter Van Roy. Overcoming software fragility with interacting feedback loops and reversible phase transitions. In *Proceedings of International conference on Visions of Computer Science*, 2008.

BIBLIOGRAPHY

- [70] Farnoush Banaei-Kashani and Cyrus Shahabi. Criticality-based analysis and design of unstructured peer-to-peer networks as "complex systems". In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid, CCGRID '03*, pages 351–, Washington, DC, USA, 2003. IEEE Computer Society.
- [71] I. Scholtes, J. Botev, A. Hohfeld, H. Schloss, and M. Esch. Awareness-driven phase transitions in very large scale distributed systems. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, pages 25–34, Oct 2008.
- [72] A. Barrat and M. Weigt. On the properties of small-world networks. *The European Physical Journal B*, 13:547–560, 2000.
- [73] C. P. Herrero. Ising model in small-world networks. *Phys. Rev. E*, 65(066110), 2002.
- [74] H. Hong, B. J. Kim, and M. Y. Choi. Comment on “ising model on a small world network,”. *Phys. Rev. E*, 66(018101), 2002.
- [75] M. Kuperman and D. H. Zanette. Stochastic resonance in a model of opinion formation on small world networks. *The European Physical Journal B*, 26:387–391, 2002.
- [76] A. Pękalski. Ising model on a small world network. *Phys. Rev. E*, 64(057104), 2001.
- [77] J.-Y. Zhu and H. Zhu. Introducing small-world network effects to critical dynamics. *Phys. Rev. E*, 67, 2003.
- [78] M. Takayasu, H. Takayasu, and K. Fukuda. Dynamic phase transition observed in the internet traffic flow. *Physica A*, 277:248, 2000.
- [79] T. Ohira and R. Sawatari. Phase transitions in a computer network traffic model. *Phys. Rev. E*, 58(1):193–195, 1998.
- [80] R. V. Solé and Sergi Valverde. Information transfer and phase transitions in a model of internet traffic. *Physica A*, 289:595–605, 2001.
- [81] G. Kunzmann and A. Binzenhofer. Autonomically improving the security and robustness of structured p2p overlays. In *Systems and Networks Communications, 2006. ICSNC '06. International Conference on*, pages 18–18, Oct 2006.
- [82] Seth Gilbert and Nancy Lynch. Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.
- [83] X Xiang. Coping with structured P2P network partitions and unifications. *JCIT: Journal of Convergence Information Technology*, 6(4):25–33, 2011.
- [84] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Puceva, and Roman Schmidt. P-Grid: A self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, September 2003.

BIBLIOGRAPHY

- [85] Anwitaman Datta and Karl Aberer. The challenges of merging two similar structured overlays: A tale of two networks. In *Proceedings of the First International Conference, and Proceedings of the Third International Conference on New Trends in Network Architectures and Services Conference on Self-Organising Systems*, IW-SOS'06/EuroNGI'06, pages 7–22, Berlin, Heidelberg, 2006. Springer-Verlag.
- [86] Anwitaman Datta. Merging intra-planetary index structures: Decentralized bootstrapping of overlays. In *Proceedings of SASO 2007, IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2007.
- [87] A. Datta. Merging ring-structured overlay indices: toward network-data transparency. *Computing*, 94(8-10):783–809, 2012.
- [88] S.M. Das, L.R. Dondeti, V. Narayanan, and R.S. Jayaram. Methods and apparatus for merging peer-to-peer overlay networks, September 2 2014. US Patent 8,825,768.
- [89] David Liben-Nowell, Hari Balakrishnan, and David Karger. Observations on the dynamic evolution of peer-to-peer networks. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 22–33, London, UK, UK, 2002. Springer-Verlag.
- [90] Ratul Mahajan, Miguel Castro, and Antony Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In M.Frans Kaashoek and Ion Stoica, editors, *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 21–32. Springer Berlin Heidelberg, 2003.
- [91] Planetlab. <https://www.planet-lab.org/>, 2007.