International Conference on Networked Systems, Netys 2016, Marrakech, Morocco

Reversible Phase Transitions in a Structured Overlay Network with Churn

Ruma R. Paul[1,2], Peter Van Roy[1], and Vladimir Vlassov[2]

May 18, 2016

[1] Université catholique de Louvain, Belgium
{ruma.paul, peter.vanroy}@uclouvain.be

[2] KTH Royal Institute of Technology, Sweden
{rrpaul, vladv}@kth.se

Introduction

- Applications are exposed to increasingly stressful environments
  - Out of data centers, to georeplication and edge computing
  - Node and communication failures are increasing as nodes increase in number
- We would like applications to survive such stressful environments and to have predictable behavior
  - We introduce the concept of Reversibility to define what survival means in arbitrarily stressful environments
  - We introduce the concept of Phase to allow applications to observe the hostility of their environment and behave accordingly
- We evaluate these concepts on a large realistic system
  - A structured overlay network with simulated environment and high churn
  - We investigate how to make it Reversible and how to build applications on top

Reversibility

Why we need Reversibility

- Suppose a distributed system running on *n* nodes providing a specific set of services
- From time *t* to *t+T*, the system experiences external stress
  - Ex. *k* nodes crash and *j* nodes join the system
  - Ex. a system partition due to a connectivity problem of the underlying physical network

- Can we ensure that the system will eventually regain its full functionality after time *t+T*?
- Does the system have a well-defined behavior during the interval [*t,t+T*]?

Reversibility (informal)

- *With Reversibility we can give affirmative answers to both questions!*

- Informally, Reversibility means that the system's functionality depends only on the current stress experienced by the system and not on the history of the stress

Reversibility (formal)

- Given a function *S(t)* that returns the system stress in some arbitrary but well-defined units
  - Ex. *S(t)* can explain how the system is partitioned as a function of time, or give churn as function of time

- A system is ***Reversible*** if there exists a function $F_{op}(id,S(t))$ of node identifier *id* and stress *S(t)* such that the set of system operations available at node *id* is $F_{op}(id,S(t))$
  - An operation is *available* for a given stress if the operation will eventually succeed (it will fail only a finite number of times if tried repeatedly and then succeed)
  - Note that when *S(t)=0* the system provides full functionality

Comparison with related concepts

- Reversibility versus Fault Tolerance
  - A fault-tolerant system is resilient for a given fault model, but its behavior outside that model is undefined
  - Reversibility is a stronger property because it guarantees that the system will recover functionality if the stress is removed
- Reversibility versus Self Stabilization
  - A self-stabilizing system survives any temporary perturbation of its internal state; it returns to a valid state when there are no perturbations
  - Reversibility is more useful in practice: it gives information about functionality even during nonzero stress

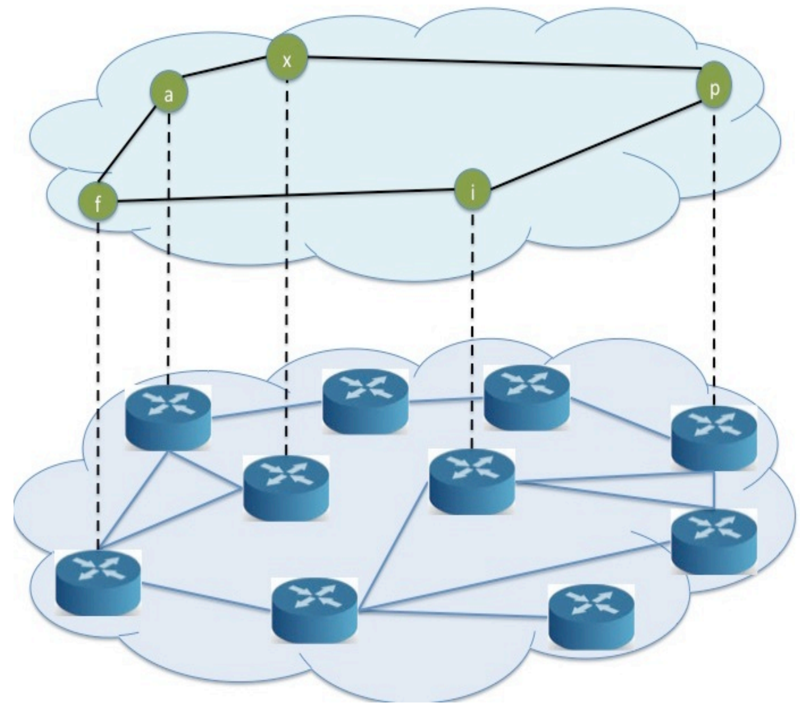
Evaluation

Evaluation

- Investigate Reversibility in the context of a realistic system
  - Representative system: a structured overlay network
- Simulated environment running on Mozart-Oz 2.0 platform
  - Simulated message delays follow Internet distribution
  - Network size of 1024 peers
- Experiments
  - First story: achieving Reversibility during high churn
  - Second story: deducing system functionality by observing structure
  - Third story: designing Reversible applications

Structured Overlay Network (SON)

- P2P Systems: Dual client/server role of each node of the system.
- Due to local cooperation of peers an overall network routing view emerges, known as an *overlay* network, on top of the underlay network.
- Structured Overlay Network: A structure is induced through the pointers maintained by each peer of the system.



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

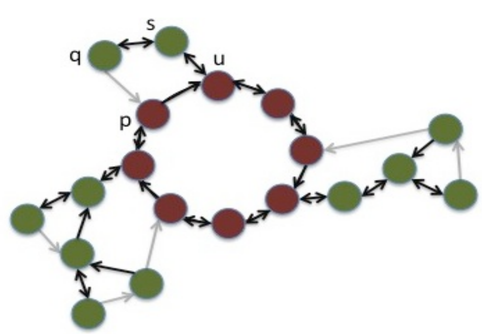
Beernet

- Beernet[3] is a representative example of the design class as per the reference architecture proposed by Aberer et.al.

- Why Beernet?
  - Similar to Chord, but with correct lock-free join operation.

  - Join/leave in Chord requires coordination of three peers that is not guaranteed due to non-transitive connectivity on Internet.
    - Non-Transitive Connectivity: *A can talk to B and B can talk to C ≠ A can talk to C.*

  - Beernet does not assume transitive connectivity. More resilient on Internet. Three step join/leave operation, each step requires coordination among only two peers (guaranteed with a point-to-point communication).
    - Consequence: Natural Branching structure. A stable **core ring** and transient **branches**.



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.

[3]B. Mejías, "Beernet: A relaxed approach to the design of scalable systems with self-managing behaviour and transactional robust storage," Ph.D. dissertation, UCL, Belgium, 2010.

Maintenance Strategies

- A Maintenance Strategy maintains correct structure of a SON
  - We investigate the Maintenance Strategies needed for Reversibility
- Several strategies are proposed in the literature:
  - Correction-on-Change/Use (used by DKS, Beernet);
  - Periodic Stabilization (used by Chord);
  - Gossip-based strategies, e.g., T-MAN (building overlay topology).
- *These strategies are complementary*
  - Correction-on-change is much more efficient than gossip, whereas gossip is much more resilient

Maintenance Strategies (cont..)

- We cover a complete space of possible maintenance strategies:

Efficiency

Maintenance Strategy	Local/ Global	Reactive/ Proactive	Fast/ Slow	Safety	Bandwidth Consumption
Correction-on-Change (for self-healing) and Correction-on-Use (provides self-optimization and self-configuration).	Local	Reactive	Fast	Yes	Small
Periodic Stabilization: correction using periodic probing.	Local	Proactive	Slow	Lookup inconsistencies and uncorrected false suspicions can be introduced	High
Overlay Merger with Passive List: Trigger Merger using falsely suspected nodes[2]	Global	Reactive	Adaptable	Yes	Adaptable
Gossip-based Maintenance, e.g., Overlay Merger[2] with Knowledge Base: Proactive approach to trigger merger using the gathered knowledge at each node.	Global	Proactive	Adaptable	Yes	Adaptable

Resiliency

. [2]T. M. Shafaat, "Partition tolerance and data consistency in structured overlay networks," Ph.D. dissertation, KTH, Sweden, 2013.

Stories and Their Contributions

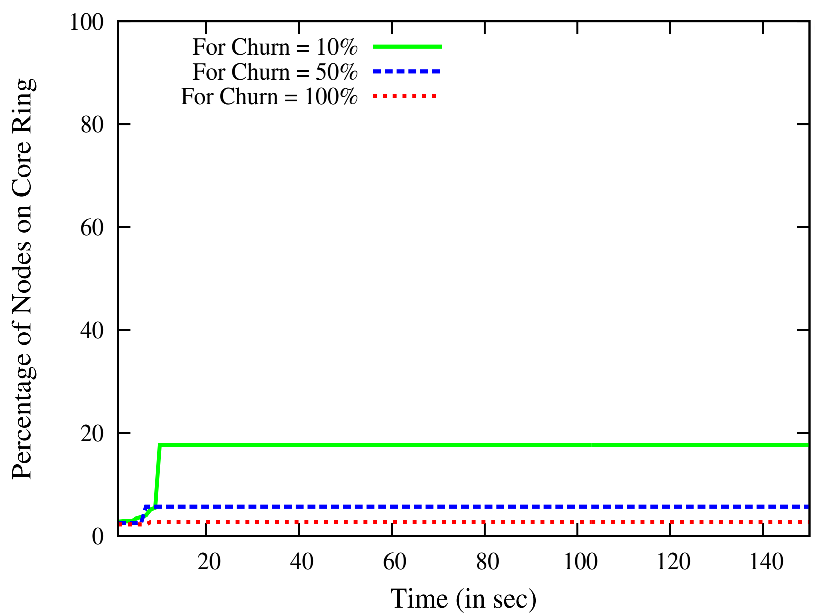
- **First Story**: "Can the system be made reversible against churn using the Maintenance Strategies?"
  - We show experimentally the need of both efficient and resilient maintenance
- **Second Story**: "Can we deduce the system's functionality by examining its structure at high churn? YES! Phase concept."
  - Insight on how to observe global structure;
  - Insight on how phase of each node is related to functionality of the system;
  - Experimental demonstration that reversible phase transitions happen in a reversible system as the stress varies
- **Third Story**: "Can we help applications to be reversible and predictable"? YES! Expose Phase of each node through an API."
  - Introduction of Phase API;
  - Insight on how the application can use phase concept to manage its behavior

First Story
Churn & Reversibility

Are the Maintenance Strategies Reversible? (1)

Churn: % of node turnover per second.
Metric: % of nodes on core ring as a function of time

Correction-on-*

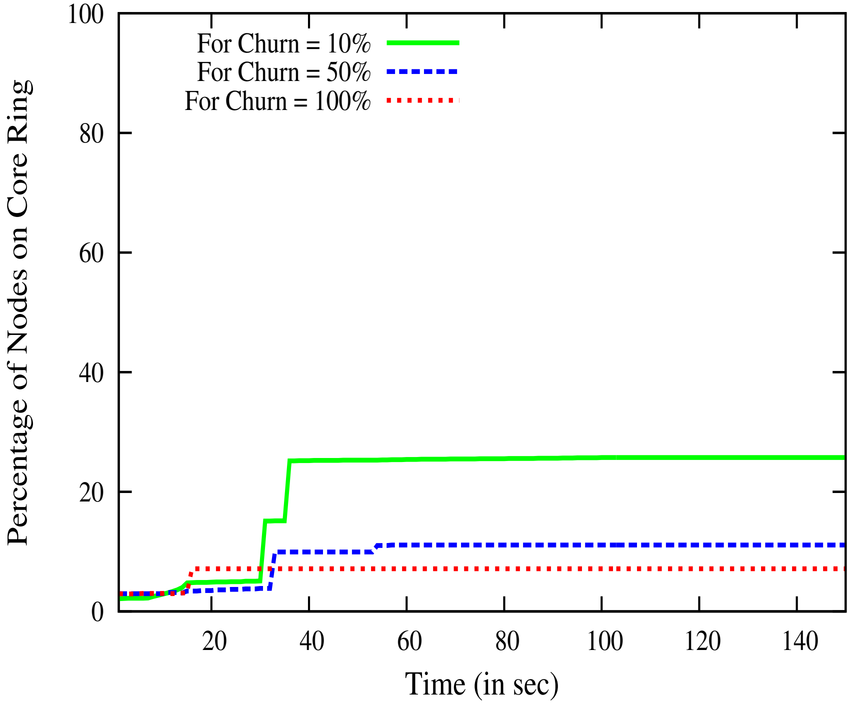


To achieve Reversibility, the percentage of nodes on the core ring should eventually approach 100%

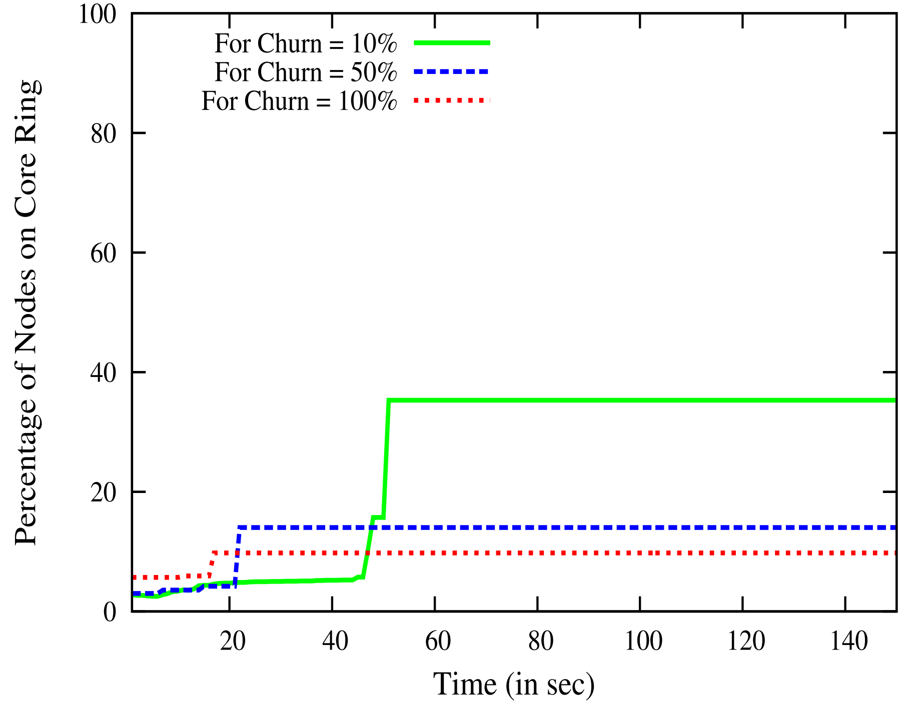
Correction-on-* is insufficient to achieve Reversibility due to lack of liveness!!

Are the Maintenance Strategies Reversible? (2)

Correction-on-* and Periodic Stabilization



Correction-on-* and Periodic Stabilization and Merger with passive list

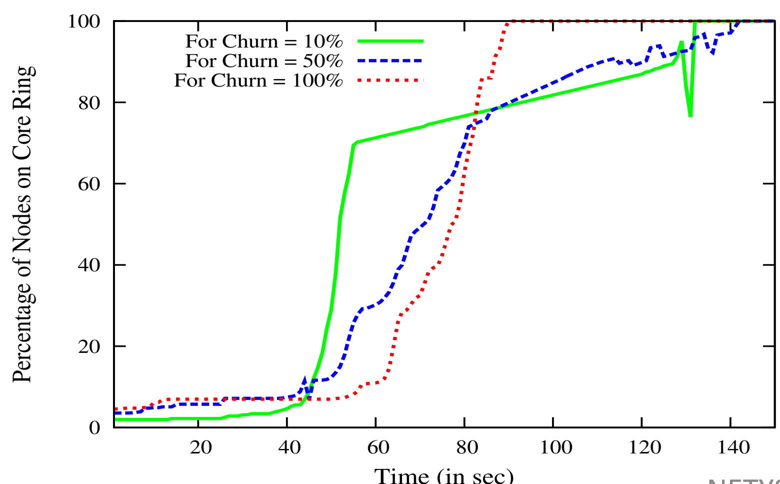


Still not Reversible. Why?

Are the Maintenance Strategies Reversible? (3)

- High churn makes overlay unstable, which does not allow new peers to complete a join
  - The churn rapidly invalidates the join reference of the new peer
- In order to make these isolated peers part of overlay, we need to re-trigger join by providing a new valid join reference.
  - **Knowledge Base** is required to get knowledge about an alive peer of overlay
- **Proactive triggering of merger using Knowledge Base** to avoid partition of the system after isolated nodes complete their join procedures.

Correction-on-*, Periodic Stabilization,
Merger with Knowledge Base.



A Perfect Ring with 100% nodes!!

Summary of First Story

- Repeated join using Knowledge Base is required to achieve Reversibility against extremely high Churn.

- Proactive merger using Knowledge Base is required to avoid partitioning of the system.

Second Story

Phase and Phase Transitions

Phase, Phase Transition & Critical Point

- System = An aggregate entity composed of a large number of interacting parts
  - Each part is a node of the SON
- A *Phase* is a subset of a system for which the qualitative properties (e.g., functional guarantees) are essentially the same
  - Different parts can be in different phases, depending on the local environment observed by the part
- Why is this interesting?
  - System functionality depends on these qualitative properties
    - Use phase for observing system functionality, but it should work without extra computation and even when communication is broken
  - Useful to applications running on top of SON in stressful environments

Phase, Phase Transition & Critical Point (Cont..)

- 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

- Reversibility and Phase:
  - Stress is a global condition that cannot be easily measured by individual nodes
  - Phase P_i at each node i is a well-defined local property
  - Phase configuration of system, $P_c = (P_1, P_2, P_3, \ldots, P_n)$.
  - The set of available operations of the system, namely $F_{det}(id, P_c(t))$.
  - *Important property:* $F_{det}(id, P_c(t))$ *approximates* $F_{op}(id, S(t))$

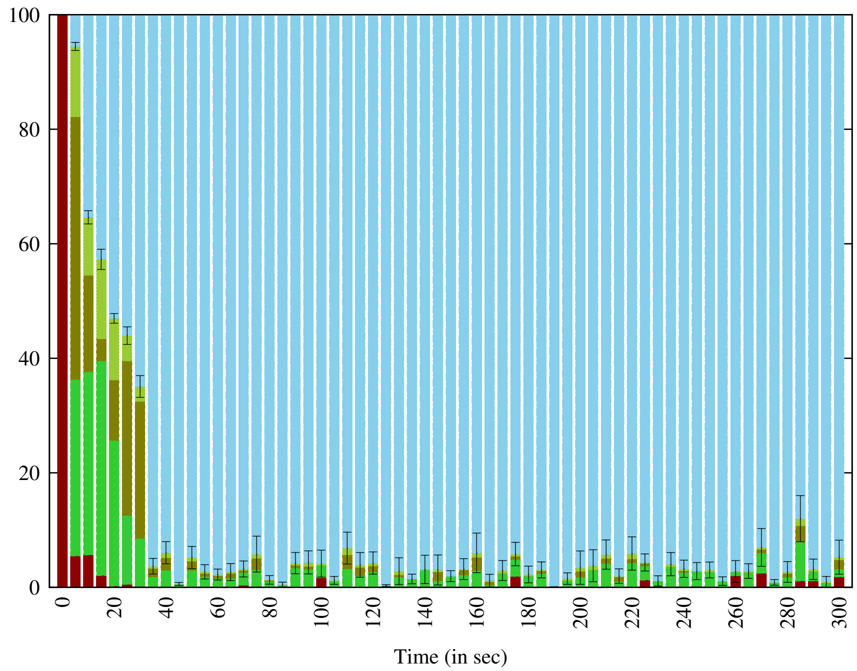
Can we observe the global structure? YES! Phase concept !!

- In case of Beernet, we can identify a qualitative property depending on neighbor behavior

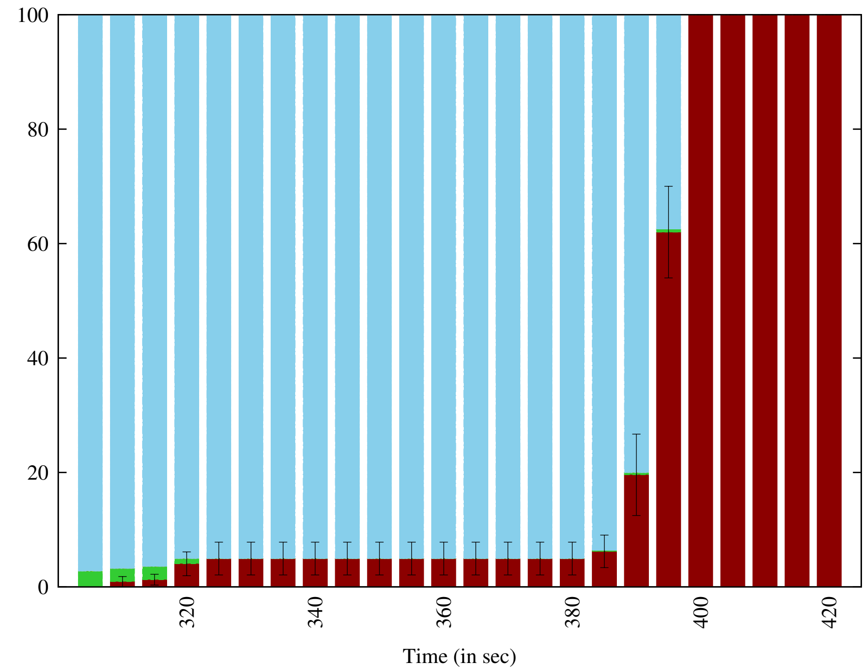
- Phases of a node are analogous to *solid*, *liquid* and *gaseous* phases in physical system (e.g., water)
  - *Solid*: neighbors do not change (core ring).
  - *Liquid*: neighbors changing (branches).
  - *Gaseous*: no neighbors (isolated nodes).

- Three liquid sub-phases in terms of available functionalities and probability of facing an immediate phase transition.
  - liquid-1: if peer is on a branch with depth <= 2 and holds a stable finger table;
  - liquid-2: if peer is on a branch with depth > 2, but not tail of a branch. The finger table holds > 50% valid fingers;
  - liquid-3: if peer is on a branch with depth > 2, and it is tail of a branch. Most fingers are invalid or crashed.

Phase Transitions in SON: red, green and blue areas correspond to % of nodes on ring (*solid*), branches (*liquid*) and isolation (*gaseous*) respectively.

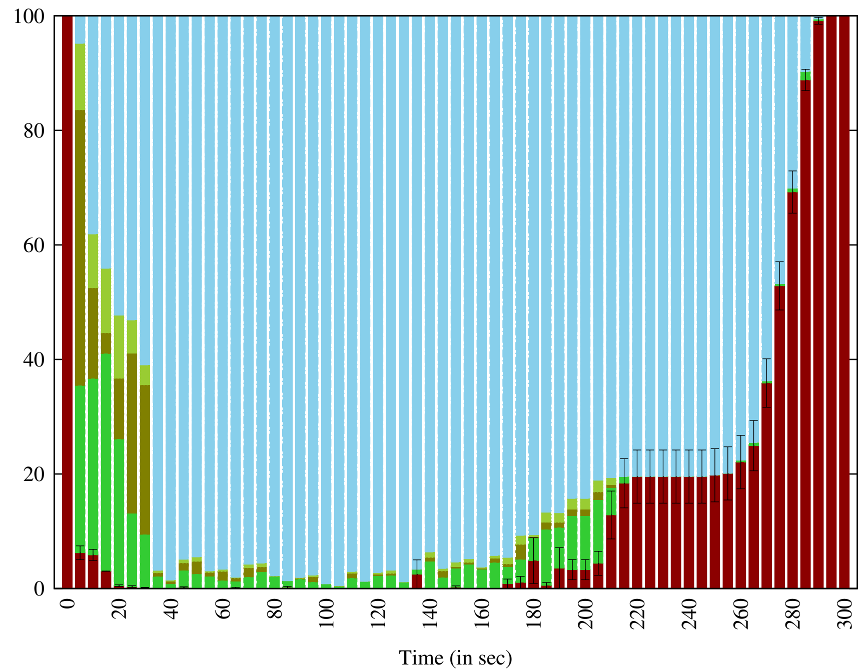


Under increasing churn during 5 minutes



After withdrawing churn

Increasing churn with time up to 100%, then decreasing churn with time:



What are Phase Transitions good for?

✓ Give useful information to the application.

✓ Can be used for efficient self-management.

Summary of Second Story

- The Phase of each node has a direct correlation with the overall functionalities (e.g., routing, availability of keys, transactions) of the system.
  - The current phase and phase transition at each node can be determined with high confidence, *without any global synchronization*.

- Reversible Phase Transitions in the system with varying stress can be observed as a by-product of making the system Reversible.
  - The system “boils” to the gaseous state (becomes disconnected) when churn increases and “condenses” from gaseous back to solid phase as churn intensity goes down.
  - Can provide useful information to the application layer using APIs.
  - Can be used for efficient self-management of the system.

Third Story
Phase API and Applications

Phase API

- An API exists on each node to expose its phase to the application layer

- Push and pull methods to communicate the current phase of a node
  - *getPhase(?P_{cur})* Binds P_{cur} to the current phase of the peer.
  - *setPhaseNotify(f)* Sets a user-defined function, *f(?P_{new})* to be executed when the phase changes. P_{new} is bound to the next phase of the peer and *f* is executed. Executions of *f* are serialized in the same thread over a stream of successive phases.

Phase-Aware Applications

- **Predictable** behavior for the users: an indicator that changes color to indicate the current phase of the underlying node.
  - Allow users to work productively offline and prevent any potential data-loss.

- **Reversibility** for the application:
  - Can increase replication factor of critical data, based on phase of underlying node;
  - Can improve throughput, by adapting philosophy of exponential back-off as TCP congestion algorithm.
  - Can manage its behavior for congestion-avoidance, thus help system to recover quickly.

- Empirical Demonstration of Phase-Aware Application design (future work)

Conclusion and Future Work

Conclusion

- In order to design provably correct decentralized networked systems, it is required to ensure their reversibility against stressful environments.
  - Build systems that are both predictable (hence, useful in practice) and reversible (hence, they survive)

- We define the concept of Reversibility to make precise what survival means in stressful environments

- We define the concept of Phase to allow applications to observe their stressful environment and act accordingly

Summary of Our Stories

- **First Story:** Repeated join and merger using Knowledge Base is required to achieve Reversibility against extremely high Churn

- **Second Story:** We observe Phases and Phase Transitions in the system as a by-product of making the system Reversible (give useful information to applications using APIs)

- **Third Story:** We introduce a Phase API to give useful information to applications and use it for phase-aware application design: predictable behavior and reversibility in the application-level semantics.

Future Work

- Continuing the work directly:
  - Deepen the analogy between phase in SONs and in physical systems;
  - Design applications that take advantage of the Phase API to survive in extremely stressful environments;
  - Gain more insights about the maintenance strategies.

- Other topics:
  - Investigate other application architectures;
  - Investigate other stresses and stress interactions;
  - Move to real environment, not simulated.

Thank You!!

Reversible Phase Transitions
in a Structured Overlay Network with Churn

Ruma R. Paul[1,2], Peter Van Roy[1], and Vladimir Vlassov[2]

[1] Université catholique de Louvain, Belgium {ruma.paul, peter.vanroy}@uclouvain.be
[2] KTH Royal Institute of Technology, Sweden {rrpaul, vladv}@kth.se