Enforcing Routing Consistency in Structured Peer-to-Peer Overlays: Should We and Could We?

Wei Chen Xuezheng Liu
Microsoft Research Asia
{weic, xueliu}@microsoft.com

ABSTRACT

In this paper, we argue that enforcing routing consistency in key-based routing (KBR) protocols can simplify P2P application design and make structured P2P overlays suitable for more applications. We define two levels of routing consistency semantics, namely weakly consistent KBR and strongly consistent KBR. We focus on an algorithm that provides strong consistency based on group membership service and weakly consistent KBR. The algorithm provides a continuum of consistency levels for applications with a tunable parameter.

1. INTRODUCTION

1.1 Why routing consistency?

The success of Internet peer-to-peer (P2P) file-sharing services has motivated considerable research on P2P systems. Among these works, one important area is structured P2P overlays that provide key-based routing (KBR) functionality [12]. KBR maps a large identifier space onto the set of nodes in the system, and it relies on the overlay topology to route any key in the identifier space to its mapped node. Previous research mainly concentrated on the design of overlay topology and KBR algorithms to improve routing performance, reduce maintenance costs, or explore the tradeoffs between routing hops and the size of the routing table. However, to make KBR a valuable building block for many P2P applications, it is critical for KBR to provide routing consistency guarantee.

Routing consistency in structured P2P overlays is the property ensuring that routings with any given key always reach the correct node mapped by the key (a.k.a the owner of the key). Unfortunately, most existing protocols only provide best-effort routing and do not guarantee this property. As a result, routings are sometimes erroneous. These routing errors become more frequent when churns and failures drive routing tables of nodes into inconsistent states. The technical report version of Bamboo [24] shows that some routing errors are difficult to correct and may exist for a long time.

Routing errors may decrease the performance of KBR-based applications or cause application errors. For example, applications (e.g. [11, 15]) using distributed hash tables to store key-value pairs may falsely report a stored key to be lost when routing to a wrong node, or start unnecessary replication maintenance. Similar situations can be found in other KBR-based applications and algorithms, such as publish/subscribe applications (e.g. [4, 6]) and P2P locking algorithm [18]. It is difficult for every individual applications to build complex distributed data structures and systems on top of an inconsistent and error-prone routing layer. To a certain extent, this makes structured P2P overlays less competent as a widely applicable building block for distributed systems.

1.2 How to enforce routing consistency?

Our research on enforcing routing consistency is mainly inspired from group membership and group communication systems (e.g. [3, 5, 28]), which have made significant advances in supporting strong consistency in dynamic systems. These systems, however, are only appropriate for cluster environments and are not scalable to large scale and more dynamic P2P environments.

If we look at KBR routing consistency as a continuous spectrum, existing KBR protocols are at the weakest end since they are best-effort and does not provide consistency guarantee, while the traditional group membership protocols are at the strongest end, because they maintain a consistent view over entire membership and KBR is reduced to one-hop membership lookup. Both extremes have their own drawbacks: the weakest end has no consistency guarantee desired by applications while the strongest end is not scalable. Therefore, our task is to balance the tradeoff between consistency and scalability and locate appropriate points in the middle of the spectrum for applications.

In this paper, we tackle the above problem by rigorously specifying and enhancing routing consistency in structured P2P overlays. We define two levels of routing consistency, namely weakly consistent KBR that eventually achieves routing consistency, and strongly consistent KBR that provides consistency guarantees even before the system is stable. Based on our specifications, designers of P2P applications can clearly understand and avoid misuse of routing functionality, as well as make formal proofs on their algorithms.

Using group membership services and weakly consistent KBR, We design a new routing algorithm that implements strongly consistent KBR. This is the first KBR algorithm that provides strong consistency guarantee as well as reasonable scalability. The algorithm provides a continuum of consistency levels with a tunable parameter, with which applications can select the appropriate consistency level and avoid complicated designs to tolerate routing errors. Our work makes KBR suitable as a general building block for many applications and potentially broadens the usage of structured P2P overlays.

1.3 Related Work

Few studies were found on providing strong semantics to P2P systems. Lynch, *et.al.* [20] studied data access atomicity in distributed hash tables (DHT), which shares the same motivation of providing stronger guarantees to P2P systems. Our paper is on KBR, which is at a lower layer and is a key component of DHT. To the best of our knowledge, this is the

first work that directly study the strong consistency semantics of KBR.

Consistency semantics have been extensively studied in group membership services (GMS) and group communication systems (see survey [10]). GMS implementations can be found in group communication systems (e.g. [3, 5, 28]) and there are several attempts to formally specify group memberships (e.g. [22, 25]). However, the specifications differ from each other due to their different focuses.

Many KBR protocols in structured overlays (e.g., [23, 26, 27, 29]) exist, but they do not focus on routing consistency guarantees. The closest one to provide some guarantee is Chord [17, 27], which supports the weak consistency semantic proposed in this paper when enhanced with an extra mechanism to detect and remove the loopy state. A separate study [9] is conducted by our group on improving these KBR protocols to guarantee weak consistency and achieve fast convergence to the consistent steady state.

The rest of the paper is organized as follows. In Section 2, we provide the formal specification of weakly consistent KBR and strongly consistent KBR with related definitions on system model and group membership service. In Section 3, we present the basic zone-based algorithm that implements strongly consistent KBR using group membership and a weakly consistent KBR primitive. In Section 4, we discuss how to make the algorithm scalable and adaptive. We conclude in Section 5 and discuss several research directions that would complete this research.

2. KEY-BASED ROUTING SPECIFICATION

2.1 System model

We consider a peer-to-peer system consisting of nodes (peers) drawn from the set $\Sigma = \{x_1, x_2, x_3, \ldots\}$. We treat time as discrete with the range $\mathcal{T}$. Nodes may join or leave the system or may crash at any time. We treat node leaves and crashes as the same class of unpredictable and unnotified events. A *membership pattern* is a function Π from $\mathcal{T}$ to 2^{Σ}, such that $\Pi(t)$ denotes the finite set of nodes in the system at time t. A membership pattern Π is *eventually stable* if there is a time t_0 such that $\Pi(t)$ does not change for all $t \geq t_0$. If Π is eventually stable, let $sset(\Pi)$ be the set of nodes that are eventually alive in the system.

Nodes communicate by asynchronous message passing over communication links. Messages sent are assumed to be unique. Messages cannot be created or duplicated by the links, and they are reliable in the sense that if both the sender and the receiver keep alive after the send event of a message, then the message will be received by the receiver eventually.

The system is *eventually stable* if there is a time after which the membership pattern is stable, and there is a link between any pair of nodes remaining in the system, and all messages can be delivered to the recipients within a certain time bound. Eventual stability of the system is only needed to guarantee liveness properties of the specifications. The assumption of the complete connectivity in the eventually stable system is for the simplicity to illustrate the main results of the paper. As discussed in Section 5, based on some previous results this assumption can be weakened and we are working on the details of a weaker and more practical network model for P2P systems to support consistent KBR.

2.2 Group membership service

For the purpose of supporting KBR, we present group membership service (GMS) as a query interface to membership views and the queries are totally order together with membership change events. This total ordering provides causality between different queries, which is usually not required in other GMS specifications. The description of GMS is kept informal due to the space constraint.

GMS maintains a *membership view* $v = (set, ver)$ where $v.set \subset \Sigma$ is a finite set of nodes and $v.ver$ is a non-negative integer indicating the version of the view. A node in the system queries the current membership view by invoking the interface action getCV(). Action getCV() always has a return value, which is either a valid view or $\perp$ indicating that the node is not yet in the view or the query failed. The membership view is updated by two internal actions join(x) and remove(x). Action join(x) is initiated by a new node x joining the system and it adds node x into the membership view, while action remove(x) is initiated by nodes that detect the failure of node x and it removes x from the view. Both actions also increment the view number. GMS totally orders all getCV(), join() and remove() actions to provide consistency guarantee on the view membership. Important properties that GMS satisfies include (but may not be limited to):

- *Causality Consistency*: If node x_1 and x_2 each invokes a getCV(), and the return of getCV() on x_1 causally precedes the invocation of getCV() on x_2, and the return values are two views v_1 and v_2, respectively, then $v_1.ver \leq v_2.ver$.
- *Agreement*: For any two views v and w returned by getCV()'s, if $v.ver = w.ver$, then $v.set = w.set$.
- *Eventual Convergence*: If membership pattern Π is eventually stable, then there is a view v such that $v.set = sset(\Pi)$ and for any node $x \in sset(\Pi)$, there is a time t_1 such that if x invokes getCV() after time t_1, the return value is v.

Causality Consistency is a safety property ensuring that the causality of the query events as defined in [16] is consistent with the increment of the view numbers. Agreement is another safety property ensuring view consistency as long as version numbers agree. Eventual Convergence is the liveness property ensuring that GMS will converge to a single view that matches the live nodes in the system provided that the membership becomes stable eventually.

GMS with the above properties can be implemented using consensus [13] or causal atomic broadcast primitives [14] to totally order all actions, and use eventually perfect failure detectors [8] to ensure view convergence to the actual set of nodes remaining in the system. An implementation is underway to evaluate the proposed protocol in this paper.

2.3 Weakly consistent KBR specification

Each node $x \in \Sigma$ has a unique id $x.id$, drawn from a key space $\mathcal{K}$. When the context is clear, we use x to represent $x.id$. Weakly consistent KBR (W-KBR for short) has one primitive w-lookup(k), where k is a key value from the same space $\mathcal{K}$. A node uses this primitive to find out the node that owns the key k. In large and dynamic P2P systems where a node cannot store the entire membership list of the system, w-lookup() is typically implemented by multihop routing.

The w-lookup() primitive either returns a $\perp$ value indicating the failure of the lookup, or a node x (including its id $x.id$ and its physical address $x.address$ for future communication).

Informally, W-KBR means that routings are eventually consistent when the system is stable for a long enough period, but they may not be consistent when the system is not stable. More rigorously, it needs to satisfy the following properties.[1]

- *Eventual Progress*: If membership pattern Π is eventually stable, then there is a time t_1 such that for any key $k \in \mathcal{K}$, if a node $x \in sset(\Pi)$ invokes w-lookup(k) after t_1, then the return value must be some $y \in sset(\Pi)$.

- *Eventual Consistency*: If membership pattern Π is eventually stable, then there is a time t_1 such that for any key $k \in \mathcal{K}$, if two nodes $x_1, x_2 \in sset(\Pi)$ invoke w-lookup(k) after time t_1 and the return values are $y_1, y_2 \in \Sigma$ respectively, then $y_1 = y_2$.

The Eventual Progress property requires that eventually all routings should successfully return a node instead of $\perp$, and the node returned should be a live node in the system. The Eventual Consistency property requires that eventually all routings with the same key will find the same node. Both properties assume that the membership is eventually stable, and together they imply that eventually every key is owned by exactly one live node in the system.

Most existing KBR protocols use best effort approach and are essentially moving toward providing W-KBR semantic, but many do not formally guarantee W-KBR. Chord protocol with loopy state removal [17] does provide W-KBR guarantee.

From a theoretical point of view, W-KBR with the above two properties can be used to implement the Ω failure detector defined in [7]. This leads to the following impossibility result, since Ω failure detector is known to be the weakest failure detector implementing Consensus, a well known problem that is impossible to implement in purely asynchronous systems with node crashes [13].

THEOREM 1. *W-KBR primitive* w-lookup(k) *for any k can be used to implement an Ω failure detector. As a result, W-KBR cannot be implemented in a purely asynchronous systems subject to node crashes (even if we do not consider node joins).*

The proof is omitted due to the space constraint. The implication of the theorem is similar to the implication of the impossibility of Consensus: The purely asynchronous system model needs to be augmented to make W-KBR implementable. In this paper, we make a simple assumption that the system is eventually stable so that W-KBR (as well as the strongly consistent KBR proposed in the next section) is implementable. Some research work studied the minimal synchrony required for implementing Ω failure detector [1, 2, 21], and these results can be potentially adapted for W-KBR. This is one of our current research topics.

2.4 Strongly consistent KBR specification

Intuitively, strongly consistent KBR (S-KBR for short) should guarantee that routings with the same key always reach the same destination no matter where the routing is started. This, however, has to be modified since the destination may changes overtime due to node leaves and joins. To deal with changes, we add a version number to the routing results. Informally, the version number tells that the returned destination is the owner of the key during this version. The version number increases when the owner of the key changes overtime.

More specifically, S-KBR uses routing primitive s-lookup(k), which returns either $\perp$ or $(x, kver)$, where x is a node and $kver$ is a non-negative integer. S-KBR needs to satisfy the following properties.

- *Causality Consistency*: If two nodes x_1 and x_2 invoke s-lookup(k) and get return values $(y_1, kver_1)$ and $(y_2, kver_2)$ respectively, and the return of x_1's invocation causally precedes the x_2's invocation, then $kver_1 \leq kver_2$.

- *Strong Consistency*: If two nodes x_1 and x_2 invoke s-lookup(k) and receive return values $(y_1, kver_1)$ and $(y_2, kver_2)$ respectively, and $kver_1 = kver_2$, then $y_1 = y_2$.

- *Eventual Stability*: If membership pattern Π is eventually stable, then there is a time t_1 such that for every $k \in \mathcal{K}$, there is a version number m_k, for every node $x \in sset(\Pi)$, if x invokes s-lookup(k) after time t_1, the return values must be non-$\perp$, and the version number in the return value is m_k.

Causality Consistency requires that the increment of version numbers is consistent with causality. Strong Consistency requires that as long as the two routings of the same key have the same version number, they will have the same destination. This property is meant to hold at all times, which is different from the Eventual Consistency property of W-KBR. One may argue that an implementation can get around the Strong Consistency property by returning different version numbers for every return values or returning failures. This, however, is constrained by the Eventual Stability property that does not allow indefinite increments of version numbers or indefinite failure returns if the membership is eventually stable. Since the implementation does not know when the system is stable, it has to try to reach strong consistency at all times.

The above properties can be matched with properties of GMS in Section 2.2, which indicates that the two problems are related and using GMS to implement S-KBR is a natural choice.

3. ALGORITHM FOR S-KBR

S-KBR can be simply built on top of a global GMS: the routing source obtains a global membership view and then calculates the destination based on the key and the membership view. Causality, consistency and liveness properties are guaranteed by GMS. The key issue, however, is that GMS is not scalable enough to handle the scale and dynamic changes of P2P systems. The way to deal with the issue is to partition the nodes into multiple components, each of which is managed by a separate GMS.

For simplicity, we restrict the key space to be a one dimensional circular space on integers from 0 to $2^n - 1$ for some integer n, similar as in [26, 27]. The space is statically partitioned into a number of *zones*, $(0, n_1], (n_1, n_2], \ldots, (n_t, 0]$.

[1] In this paper, we omit other KBR properties such as load balance and focus on routing consistency properties.

On node x:

```
1  To execute s-lookup(k):
2    y ← w-lookup(k);
3    if y = ⊥ or y ∉ Z(k) then return ⊥;
4    send message (SLOOKUP, k) to y;
5    wait until [received (SLOOKUPACK, *) or timed out];
6    if received (SLOOKUPACK, z, kver) then
7      return (z, kver) else return ⊥;
8  On receive (SLOOKUP, k) from a node y:
9    v ← getCV();
10   if v ≠ ⊥ then
11     z ← selectNode(v.set, k);
                          {select a node from the current view}
12     send (SLOOKUPACK, z, v.ver) to y;
```

Figure 1: Algorithm implementing S-KBR

We denote Z as a zone partition of the key space $\mathcal{K}$, and for any key k, $Z(k)$ represents the zone that covers the key k. Nodes whose ids fall into the same zone form a group and are managed by a GMS for the zone. Zone size is a tunable parameter that controls the tradeoff between scalability and consistency, and we will discuss it in Section 4.1.

Figure 1 shows the S-KBR algorithm, which has two phases. In the first phase, the key k is routed to a node in the target zone $Z(k)$. This routing can be done by a variant of W-KBR, which requires that eventually routings with key k always fall into the target zone $Z(k)$ (stronger than the Eventual Progress property of W-KBR), but they do not necessarily end up in the same node (weaker than the Eventual Consistency property of W-KBR). As a result, we need to require that eventually every zone contains some node in the system, which is discussed in Section 4. Existing KBR protocols are usually sufficient to be used as such a W-KBR variant in practice. In the second phase of routing, once a key is routed into a node within the target zone, the node queries the zone's GMS to retrieve the current view of the zone. Based on the view and the key, the node calculates the destination node and returns it with the version number of the zone as the routing result. If a node within the target zone cannot be located in the first phase, the routing returns failure. This is the situation where the algorithm chooses to sacrifice liveness to maintain strong routing consistency.

Since each zone has its own GMS to manage the membership within the zone, there is an issue on how a new node joining the system locates its zone's GMS. This bootstrap problem can be solved by a separate bootstrap service that connects nodes with its GMS, or it can be solved by the same W-KBR variant we used in the first phase of S-KBR routing. When a node is joining the system, it uses its own id as the routing key and uses the W-KBR variant to route its own id to a contact node in its zone. Since the W-KBR variant guarantees that eventually it will route the key to a node in the same zone, this bootstrap will be successful eventually.

We proved the correctness of the algorithm but omit the proof due to the space constraint. The following theorem states its correctness.

THEOREM 2. *Under the condition that the GMS and W-KBR variant satisfy their own properties specified, algorithm in Figure 1 satisfies the Causality Consistency and Strong Consistency properties of S-KBR. Moreover, if the system is eventually stable, the algorithm also satisfies the Eventual Stability property.*

The above theorem is not in conflict with Theorem 1 because it assumes eventual stability of the system for the liveness property. The safety properties, namely Causality Consistency and Strong Consistency, do not rely on eventual stability of the system and they hold for any asynchronous systems.

The algorithm always maintains routing consistency, even when the system is not stable. To guarantee such strong consistency, the algorithm sacrifices routing liveness when the first-phase routing does not reach the target zone. An alternative is for the algorithm to return an inconsistent result, which means it sacrifices routing consistency in favor of routing liveness. This is a choice that the designer of the system can made, but for the purpose of demonstrating that strong routing consistency can be guaranteed, we choose the algorithm that sacrifices routing liveness in favor of strong routing consistency.

In the basic algorithm, the version number of any key in a zone changes as soon as the zone version changes. We can reduce key version number changes to improve routing consistency guarantee by the following two steps. First, we define a stable selectNode() function such that most keys' version numbers do not change as view changes. Second, in addition to maintaining a zone view, the GMS also maintains a data structure to record key ownership versions for each key in the zone and use it for routing return values instead of the zone version number. It is not hard to develop the details of the scheme which we omit in this paper.

4. DISCUSSION

The previous section provides the algorithm that is proven to support S-KBR. However, to make the basic algorithm applicable in dynamic P2P environments, a number of issues need to be addressed. In this section, we discuss our proposals to address these issues. Currently we are formulating the technical details of these proposals. Our next step of research is to fully implement the entire protocol stack from GMS to S-KBR, and conduct extensive analysis and simulations to validate the feasibility of our approach.

4.1 Zone size determination

Zone size is the tunable parameter that determines the tradeoff between scalability and routing liveness (or routing consistency). We assume node ids are randomly generated and we use equally-sized zones and thus each zone contains roughly the same number of nodes (unbalanced zones are dealt with in the next section).

With a larger zone size, each zone contains more nodes, and thus more first-phase routings with the W-KBR variant will fall into the target zone, leading to successful and consistent routing result. But it increases the GMS query and maintenance cost and reduces scalability. In the extreme case where the entire key space is covered by a single zone, the algorithm is reduced to the global GMS based algorithm. On the other hand, with a smaller zone size, more first-phase routings will fall outside the target zone, leading to more routing failures or, if we choose to return a result, more violations to routing consistency. The extreme is that each zone only covers one node and the algorithm is essentially

reduced to a W-KBR algorithm. Therefore, tuning the zone size provides a continuum of consistency levels from weak consistency to strong consistency. Further analysis and simulations are planned to quantify the impact of zone size on the tradeoff between scalability and routing consistency.

4.2 Zone merges and splits

System churns or system scale changes may cause some or all zones in the system become overloaded or underloaded. To key the number of nodes within each zone at the same level as determined by the consistency to scalability tradeoff, we use zone merges and splits. In particular, when a zone becomes overloaded, we split it in halves; when a zone becomes underloaded, we merge it with neighboring zones.

To maintain routing consistency, we need to maintain correct versions when the zone changes. The idea is that instead of having one static zone $Z(k)$ for a key k, we have a sequence of zones $Z_1(k), Z_2(k), \ldots$ for key k, following the causal order of merges and splits. The version numbers along the sequences of zones associated with key k is monotonically increasing. This can be achieved by always using larger version numbers for the new zones after merges or splits.

Another important issue is that zone merges have to be agreed upon all relevant zones, otherwise it may result in inconsistency in zone partition. Such agreement can be achieved by running consensus among zones, effectively enforcing consistency at a higher and inter-zone level. This leads to a hierarchical design in which a small number of centralized servers are at the top level enforcing global consistency, but they are only needed when consistency cannot be resolved at lower levels, and thus they are rarely needed. The lower levels are more decentralized and maintain local consistency at a manageable scale. Moreover, an actual implementation may choose to remove the higher levels of the hierarchy to sacrifice global consistency in some rare cases for a more decentralized and scalable solution.

4.3 Dead zones

A zone cannot make any progress and becomes dead when a majority of nodes in the zone are dead before the GMS of zone takes any action. When a zone becomes dead, we need to remove it and reactivate the zone, otherwise, any routing with a key in the zone will not be successful. This is the zone bootstrap problem, and it can be dealt with using the same hierarchical design for zone merges and splits. Each zone is monitored by a number of other zones and consensus is run among these zones for removing and reactivating a dead zone.

5. CONCLUDING REMARKS

This paper formally defines the routing consistency semantics of key-based routing in P2P systems. By utilizing prior research in group communication systems, it provides the framework and basic results to show that strongly consistent KBR is achievable. It further provides guidelines on how to make the result scalable and adaptive, and how to balance the tradeoff between scalability and consistency, so that it may be potentially suitable for practical P2P systems.

There are a number of research topics and directions we are currently pursuing. First, to provide a complete set of protocols including zone merges, splits, and dead-zone handling, we are formulating a two-level hierarchical design of the system as sketched in Section 4. The higher level of the hierarchy handles global consistency tasks such as inter-zone agreement for zone merges and zone reboot, and it is invoked less frequently. The lower level is for local consistency and uses a local group membership service. We will formally prove the correctness of the hierarchical design once it is completely formulated. Second, we plan to implement the hierarchical design using the WiDS platform [19], and utilizes the simulation and verification capabilities of WiDS to check the correctness of the design and verify the scalability of the system. Third, we are also working on weakening the system model to match more practical scenarios. On this front, there are some existing work studying the minimal synchrony requirement needed to implement Ω failure detector [1, 2, 21]. We are working on similar ideas and applying them to the P2P system context.

Overall, we believe that a scalable hierarchical design is a general design approach applicable not only to the specific key-based routing problem in P2P systems, but also to large scale and dynamic distributed systems in general. Our goal is to complete a rigorous study on the specifications and the algorithms of the hierarchical design with both analytical proofs and experimental evaluation. We believe that both the approaches and the final results will be beneficial to many areas related to distributed system research.

Acknowledgments

We would like to thank Yu Chen for his helpful discussions and thank Lidong Zhou, Ben Y. Zhao and the anonymous referees for their helpful comments that improve the presentation of the paper.

REFERENCES

[1] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing omega with weak reliability and synchrony assumptions. In *Proceedings of the 22ed ACM Symposium on Principles of Distributed Computing*, July 2003.

[2] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing*, pages 328–337, July 2004.

[3] Yair Amir, Danny Dolev, Shlomo Kramer, and Dahlia Malkhi. Transis: A communication sub-system for high availability. In *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, pages 76–84. IEEE Computer Society Press, July 1992.

[4] Roberto Baldoni, Carlo Marchetti, Antonino Virgillito, and Roman Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Proceedings of the 25th International Conference on Distributed Computing Systems*, June 2005.

[5] Kenneth P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, December 1993.

[6] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8), October 2002.

[7] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving

consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[8] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[9] Yu Chen and Wei Chen. Self-stabilizing and fast convergent structured overlay, 2005. under submission.

[10] Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: A comprehensive survey. *ACM Computing Surveys*, 33(4):427–469, December 2001.

[11] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, October 2001.

[12] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common API for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, February 2003.

[13] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[14] Vassos Hadzilacos and Sam Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report 94-1425, Department of Computer Science, Cornell University, Ithaca, New York, May 1994.

[15] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[16] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[17] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, July 2002.

[18] Shiding Lin, Qiao Lian, Ming Chen, and Zheng Zhang. A practical distributed mutual exclusion protocol in dynamic peer-to-peer systems. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems*, February 2004.

[19] Shiding Lin, Aimin Pan, Zheng Zhang, Rui Guo, and Zhenyu Guo. WiDS: An integrated toolkit for distributed system development. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Operation Systems*, June 2005.

[20] Nancy Lynch, Dahlia Malkhi, and David Ratajczak. Atomic data access in content addressable networks. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[21] Dahlia Malkhi, Florin Oprea, and Lidong Zhou. Omega meets paxos: Leader election and stability without eventual timely links. In *Proceedings of the 19th International Symposium on Distributed Computing*, September 2005.

[22] Roberto De Prisco, Alan Fekete, Nancy Lynch, and Alex Shvartsman. A dynamic view-oriented group communication service. In *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing*, pages 227–236, June 1998.

[23] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the SIGCOMM'01ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August 2001.

[24] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. Technical Report UCB-CSD-03-1299, University of California at Berkeley, December 2003.

[25] Aleta Ricciardi and Ken Birman. Process membership in asynchronous environments. Technical report, Department of Computer Science, Cornell University, April 1994.

[26] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of ACM Middleware*, November 2001.

[27] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the SIGCOMM'01ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August 2001.

[28] Robbert van Renesse, Kenneth P. Birman, and Silvano Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[29] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.