Proceedings of APCC2015 copyright © 2015 IEICE 14 SB 0087

Testaments for Resilient Structured Overlay Networks

Kimihiro Mizutani*, Takeru Inoue*, Toru Mano*, Osamu Akashi*, Satoshi Matsuura†, Kazutoshi Fujikawa‡
*NTT Network Innovation Laboratories
Email: {mizutani.kimihiro, mano.toru, inoue.takeru, akashi.osamu}@lab.ntt.co.jp
†Tokyo Institute of Technology
Email: matsuura@gsic.titech.ac.jp
‡Nara Institute of Science and Technology
Email: fujikawa@itc.naist.jp

***Abstract*—The routing efficiency of structured overlay networks depends on the consistency of *pointers* between nodes. This consistency can, however, break temporarily when some overlay nodes fail, since it takes time to repair the broken pointers in a distributed manner. Conventional solutions utilize "backpointers" to quickly know the failure among the pointing nodes, which allows them to fix the pointers in a short time. Overlay nodes are, however, required to maintain backpointers for every pointing node which incurs significant consistency check and memory overheads. This paper proposes a novel light-weight protocol; an overlay node gives a "testament" containing its acquaintances (backpointers) only to its successor (i.e., clockwise closest node), and other nodes are freed from maintaining it. Our carefully-designed protocol guarantees that all acquaintances are registered with the testament even in the presence of churn, and the successor notifies the acquaintances for the deceased. Even if the successor passes away and the testament is lost, the successor of the successor can identify the acquaintances at a high success ratio. Simulations show that our protocol greatly reduces the mean time to repair (MTTR) and memory overheads while messaging cost increases.**

I. Introduction

Several "$\log N$" structured overlay networks have been proposed such as Mercury [1], Symphony [8], and Pastry [11]; they provide an efficient routing service of just $O(\log N)$ hops with $O(\log N)$ pointers (i.e., node IDs) per node in a network of N nodes. A pointer becomes void if the pointee node fails (or irregularly leaves the network). In structured overlay networks, an overlay node periodically sends messages to its pointee nodes (e.g., Pastry extension [2]), and if it detects that a node has failed, the corresponding pointer is updated to indicate another node. A set of overlay nodes that maintain pointers to the same failed node is called a set of "backpointer" nodes for the failed node [15], and all of them have to update the broken pointer to ensure consistency. In the structured overlay networks, backpointer nodes try to repair their broken pointers independently, and it takes a long time for *all* backpointer nodes to complete the repair.

Cooperative keep-alive protocols such as SN+BPTR [15] focus on improving the repair time for node failure and the keep-alive message overhead in those overlay. These protocols force overlay nodes to maintain a set of backpointers for their pointers (Fig. 1a). From the figure, if node j detects node i failure, it immediately notifies the backpointer nodes

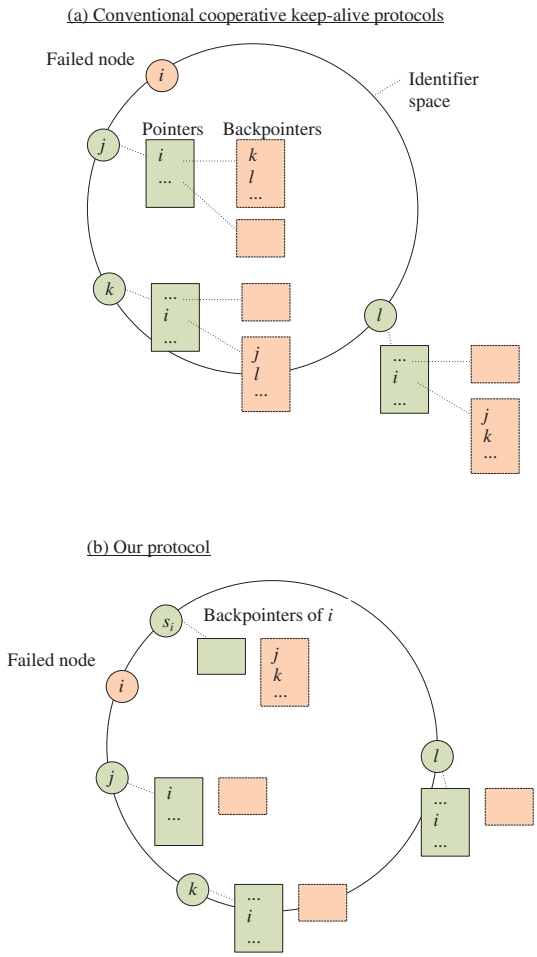

Fig. 1. (a) Conventional cooperative keep-alive protocols. (b)Our protocol.

(k and l) of the i's failure. Comparing the time taken to repair all broken pointers with plain overlays, the conventional cooperative keep-alive protocols greatly reduce the time if their keep-alive messaging frequencies are the same values. This is because the repair process completes just after the first nodes detect the failure. However, maintaining a set of backpointers for all pointer nodes imposes heavy memory

overheads on overlay nodes, i.e., $O(\log^2 N)$ states per node. In addition, $O(\log N)$ backpointee nodes must be updated through periodically keep-alive messaging after a single node joins or leaves.

This paper proposes another cooperative keep-alive protocol that relies on only a single set of backpointers per node (Fig. 1b) in the mentioned periodic pointer management overlays. In our protocol, backpointers are maintained by just the successor. From the figure, if node j detects i's failure, node j notifies the successor s_i of the failure, and then node s_i forwards it to i's backpointers. Our protocol repairs the broken pointers just after the first detection, and only $O(\log N)$ states need be held. Backpointers are kept updated so as not to miss any pointing node, since our protocol is designed to preserve invariance between a pointer and backpointer. Even if the successor fails as well and the backpointers are lost, they can be estimated from identifier shifts, which is described in III-C.

Our protocol is simple but effective. It is evaluated in comprehensive simulations extending Pastry [2]. Our protocol notifies all backpointer nodes rapidly compared to the conventional cooperative keep-alive protocol. The memory overhead is reduced by several orders of magnitude compared to the conventional protocol, and the messaging cost is only 8% larger than the conventional protocol.

II. Related Work

SN(P)+BPTR [15] utilizes a pointer node to find the backpointers ; e.g., in Fig. 1a, node i maintains its backpointers (nodes j–l), and tells them themselves. Let I be the keep-alive interval used in plain overlay networks and let K be the base of the overlay network (e.g., $K = 2$ in Mercury [1] and Chord [13]), these conventional cooperative keep-alive protocols require roughly the time of $\frac{I}{\log_K N}$ to notify all backpointer nodes on average, while plain overlay networks need I. However, each overlay node has to maintain backpointers of every pointer, i.e., up to $O(\log^2 N)$ entries, since each of $O(\log N)$ pointers has $O(\log N)$ backpointers. This overhead undermines the primary advantage of structured overlay networks; i.e., $O(\log N)$ hop routing is realized only with $O(\log N)$ states per node. In our protocol, each node maintains backpointers of predecessor only (Fig. 1b), so the state amount is $O(\log N)$.

SN(P) is also proposed in reference [15] and it does not maintain backpointer states. In SN(P), a node notifies pointee node failure to all pointee nodes, not backpointee nodes. Therefore, the detection time is less improved than that of SN(P)+BPTR. CKA [4] utilizes backpointers, but it is aimed at reducing the number of messages in pointer repair, not accelerating repair times. Reference [9] assumes multiple overlay networks are established on a single set of nodes, and node failures are shared between the networks.

References [3], [6], [10], [12], [14] predict node failures based on statistical failure models such as Weibull distribution; they can reduce the number of keep-alive messages without increasing failure detection delay. These techniques can be integrated into our protocol to improve its performance.

Reference [5] employs a death certificate so as not to resurrect a deleted item in distributed systems. This technique ensures that the deleted item is never overwritten by an older version, but it does not accelerate delete notification.

III. Proposed Protocol

This section details our proposes. Section III-A defines our network model. Section III-B describes backpointer maintenance routines, and Section III-C explains pointer repairs in the case of node failure.

A. Network Model

Overlay nodes have their own IDs. The ID space is assumed to be cyclically numbered and so we have to take a "modulo" by the space size to avoid overflow, but we omit the modulo operation in this paper for simplicity. The successor of node $i \in [1, N]$ is denoted by s_i, while the predecessor is denoted by p_i.

The set of pointers maintained at node i is denoted by P_i, while the set of backpointers for node i is denoted by B_i; e.g., in Fig. 1b, $P_j = \{i, \cdots\}$ and $B_i = \{j, k, \cdots\}$. Clearly, we have,

$$i \in P_j \iff j \in B_i, \tag{1}$$

if P_j and B_i are correctly updated. In all structured overlay networks, we can assume $s_i \in P_i$. In P of our protocol, each entry is associated with the addresses of the corresponding pointer node and its successor. Similarly in B, each entry is associated with the address of the corresponding backpointer node. B_i^k indicates B_i maintained by node k ($i \neq k$).

Node j periodically sends keep-alive messages to node $i \in P_j$ to maintain pointers and backpointers, as will be described in Section III-B. Upon failing to receive any response from node i within a specified waiting period, node j investigates node i's state using several messages. If node j determines node i has left the network, node j begins the pointer repair process, as is described in Section III-C. We assume that overlay nodes try several times to resend a message if a response has been timed out. If node n joins, node n begins the pointer update process described in Section III-B.

B. Backpointer Maintenance

Node i maintains a set of pointers, P_i, a set of backpointers, B_i, and a set of backpointers of predecessor, $B_{p_i}^i$; B_i is used for quick backpointer repairs in the case of successor failure, though it is omitted in Fig. 1b[1]. This subsection discusses a maintenance protocol of $B_{p_i}^i$ (or $B_i^{s_i}$ below).

As noted as (1), consistent pointers and backpointers must be $i \in P_j \iff j \in B_i^{s_i}$. However, it is difficult to keep this invariant in distributed systems, and so in our protocol it is relaxed to,

$$i \in P_j \Rightarrow j \in B_i^{s_i}. \tag{2}$$

This relaxed invariant is enough to repair all broken pointers, since all pointers have the corresponding backpointers in (2). The converse is not true, and so some backpointers can be invalid, which merely raises some ineffective notification messages.

[1]Note that in conventional cooperative keep-alive protocols, sets of backpointers maintained by node i are given by $\{B_j^i : j \in P_i\}$.

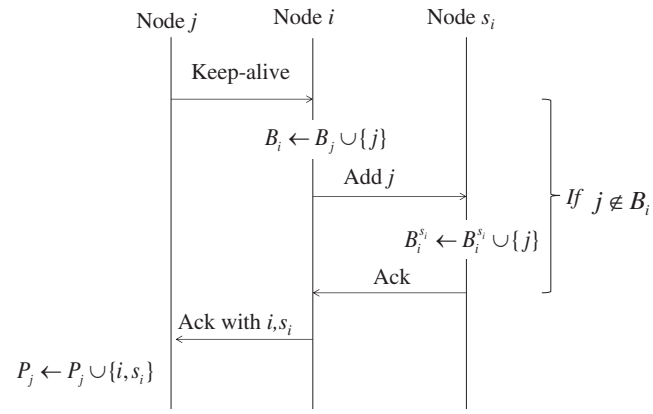

Fig. 2. Backpointer maintenance routine in first keep-alive.

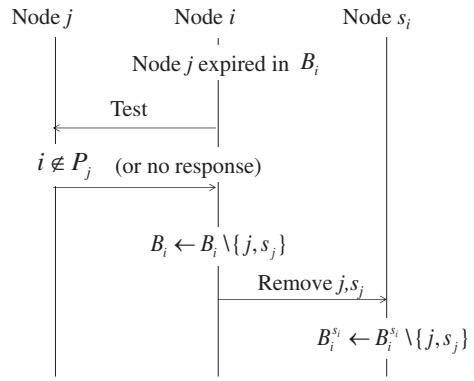

Fig. 3. Backpointer maintenance on keep-alive expiration.

Fig. 2 shows the backpointer maintenance protocol. Node i receives a keep-alive message from node j. If j is not included in i's backpointer set, i.e., $j \notin B_i$, node i adds j to B_i and then asks the successor, s_i, to add j to $B_i^{s_i}$. Upon receiving an ack from i, node j updates P_j by i associated with s_i if i is not included[2]. In this protocol, pointer i is never added to P_j unless j is not in $B_i^{s_i}$, and so (2) holds.

If node i has not received at least one keep-alive from j in a specified period, j is judged to have expired (Fig. 3). Node i, then, asks j whether the corresponding pointer is valid. If not (or no response received), j is removed from B_i. Ditto s_i. Invariant (2) is still satisfied in this case. When node n joins and finds all pointers by employed routing algorithm, node n gets the B_i from the predecessor i and notifies i of own join. Then, node i notifies both backpointer nodes listed in B_i and the predecessor p_i of updating s_i by n (Fig. 4). At the same time, node i keeps successor node s_i of n. When node i detects own successor s_i failure, the node i replicates own backpointers to the successor node s_{s_i} of s_i. Then, node i gets the successor node $s_{s_{s_i}}$ of s_{s_i} from s_{s_i}. Finally, node i updates own successor in both B_i and the predecessor by s_{s_i}(Fig. 5).

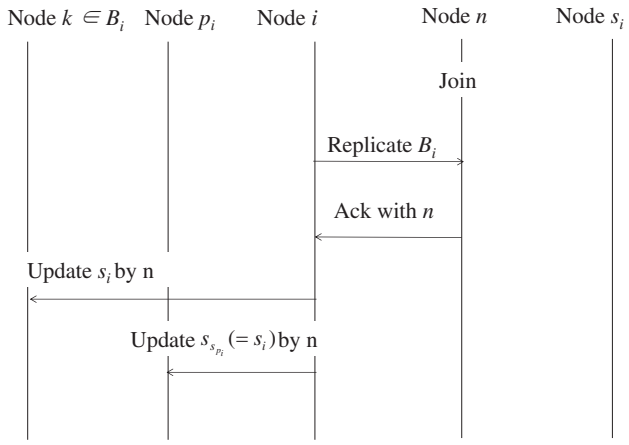

Fig. 4. Pointer update on successor join.

Invariant (2) can break in a short time just after node i fails. When node j detects i's failure(not successor's failure), node j notifies the failure by pointer repair process. The critical period is quite short since node i is quickly informed by our pointer repair process.

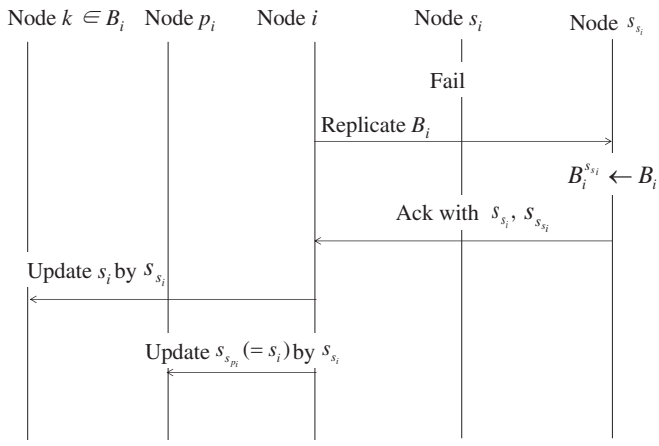

Fig. 5. Backpointer maintenance on successor failure.

C. Pointer Repairs

This section begins with the case of (2), that is, outside the critical period; we then dive into the critical period in detail. Assume that node i fails and node j detects the failure (Fig. 6). Node j first removes i from the pointer set, P_j, and notifies s_i of i's failure (node j maintains s_i's address since $i \in P_j$). Since s_i is a substitute for i, s_i uses B_i as B_{s_i} and replicates it on s_{s_i}. Finally, node s_i asks nodes j and k's to remove i from their pointers and to add s_i to their sets of pointers. Throughout the repair process, (2) is satisfied.

[2]In Fig. 2, if node i detects s_i's failure in adding j to s_i, node i interrupts the join process and updates the successor (Fig. 5). The join process is then resumed. The impact of this is evaluated in the experiments

If node i fails in the critical period (i.e., nodes i and s_i fail almost simultaneously, as shown in Fig. 7), detector j notifies s_{s_i} of i's failure; the notification is destined for $\max(s_i) + 1$ and is delivered to s_{s_i} by overlay routing, where $\max(s_i)$ is

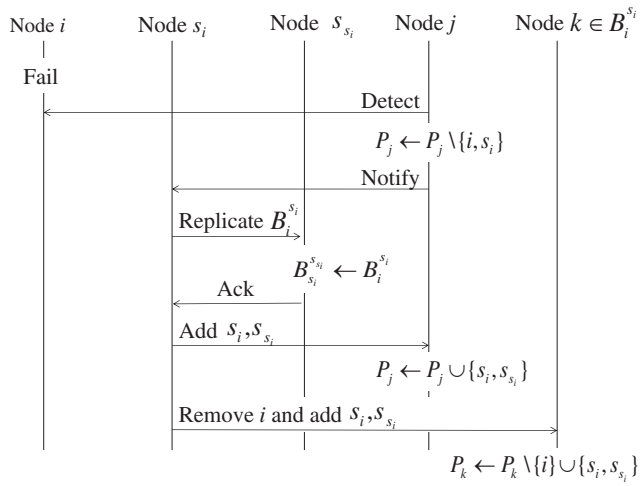

Fig. 6. Pointer repairs with backpointers.

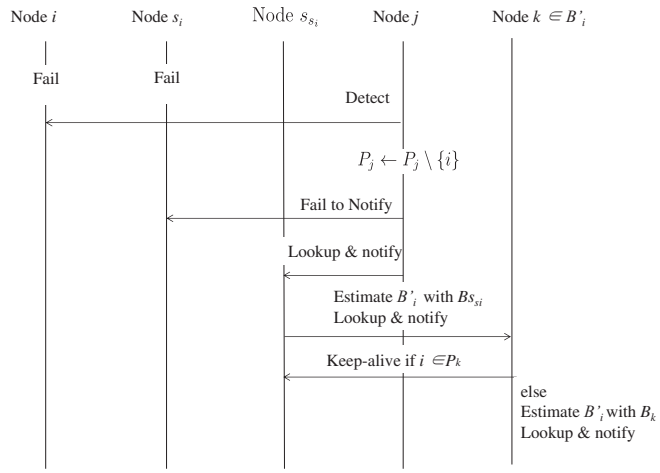

Fig. 7. Pointer repairs with backpointers under successive node failures; the dashed lines indicate that messages are delivered by overlay routing with $O(\log_K N)$ hops.

the upper bound of s_i's range (e.g., $\max(s_i) = (s_i + s_{s_i})/2$ in Pastry). However, node s_{s_i} does not have B_i.

Here, we describe how node j (possibly s_{s_i}) can estimate B_i using B_j. We assume that B_i would look like B_j if the pointers are shifted by the distance between i and j, $d(i, j)$. The definition of $d(i, j)$ differs according to the routing algorithms. In Pastry and Chord [13], the distance $d(i, j)$ means $|i-j|$. Given the distance, a set of estimated backpointers, B'_i, is determined by shifting them,

$$B'_i = \{suc(d(i, j) + k) : k \in B_j\},$$

where $suc(*)$ is the successor node of the argument. Instead of node s_i, node s_{s_i} finds $k \in B'_i$ by overlay routing, e.g., querying `get((d(`s_{s_i}`,`i`)+l))` $l \in B_{s_{si}}$. Then, s_{s_i} notifies $k \in B'_i$ of i's failure. If the estimation succeeds $i \in P_k$, node k runs the keep-alive routine of Fig. 2 with s_{s_i} (the routine is omitted in Fig. 7). After the repair process, (2) is restored by the keep-alive function. However, $B_i\prime$ does not correspond exactly to B_i in general and estimation can fail. In this case, the estimation notification should be repeated by k to notify all the nodes in B_i correctly.

Though we control the repeated notification process simply by using time-to-live in the experiments, sophisticated selective information dissemination technique [7] can be applied to control the repeated notification. Unlike [4], our protocol employs the repeated notification only if a node fails in the critical period. There can be corner cases where our protocol does not work In such a corner case, pointers cannot be repaired soon, but eventually fixed as plain structured overlay networks do.

IV. Experiments

Our protocol was evaluated by message level simulation through original simulator. It was implemented based on Pastry-extension($b = 4$) [2]. Our protocol was compared with a conventional keep-alive protocol (SN+BPTR($c = 3, k = 1, T_{boost} = 3$) [15]) as well as plain in Pastry. The keep-alive interval of node i was set at $I \cdot |B_i|$ while the other protocols use $I \cdot \log N$. With this setting, the number of keep-alive messaging per second are approximately the same in all protocols (we note that this keep-alive interval setting was accepted in [15]). ID space was $[0, 2^{20} - 1]$. Simulation was conducted for 86,400 time units for each plot. The other parameters are presented in Table I. In the simulation, overlay nodes joined and left the network with probability $1/L$, and the network included N nodes on average throughout the simulation. A node can recognize a node failure by three keep-alive message timeouts or receipt of node failure notification.

In Fig. 8, our protocol and the conventional protocol outperformed the plain protocol. Our protocol had smaller MTTRs than the conventional protocol even though it requires repeated notification in successive node failures (Fig. 7).

Fig. 9 shows the probability of critical period occupancy; it indicates that critical period occurred frequently with short lifetimes, but the probability is considerably low.

Fig. 10 shows CDF of the number of pointers and backpointers maintained per node. The conventional cooperative keep-alive protocol maintained many more backpointers than our protocol. In particular, the most loaded node had 10 times as many backpointers as our protocol. Fig. 11 shows the total number of pointers and backpointers conveyed in messages. Our protocol updated the backpointers stored on successor node only when a node joins or leaves. On the other hands, $O(\log N)$ backpointer nodes had to be updated through keep-alive messaging after a node joins or leaves in conventional protocol. For these reasons, the conventional protocol conveyed 10 times as many backpointers as our protocol.

Fig. 12 shows the sum of the number of messages for all nodes, including all maintenance messages like neighbor discovery, keep-alive, and failure notification. Our protocol used only slightly more messages than the conventional protocol despite the use of repeated notification because the probability of the critical period occupancy was low(Fig. 9).

TABLE I. PARAMETERS USED IN SIMULATION

K	2
N	1000
L	[100, 1000]
I	10
T	(0, 2); range of uniform distribution
T_{out}	4
D	3
E	5

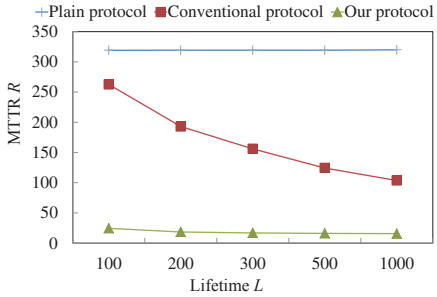

Fig. 8. Mean repair times of Pastry

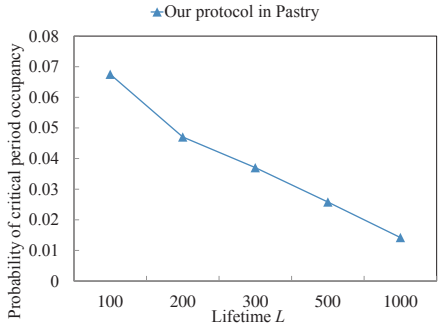

Fig. 9. Probability of critical period occupancy in lifetime; N = 1000

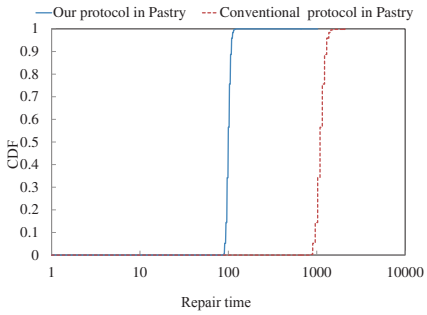

Fig. 10. CDF(Cumulative Distribution Function) of pointers and backpointers; N = 1000 and L = 1000.

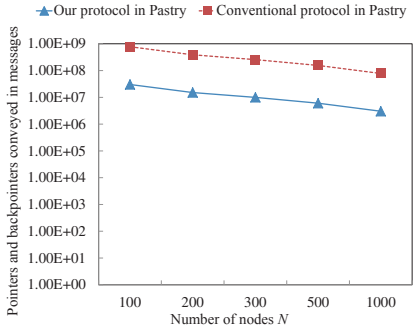

Fig. 11. The total number of pointers and backpointers conveyed in messages; L = 1000.

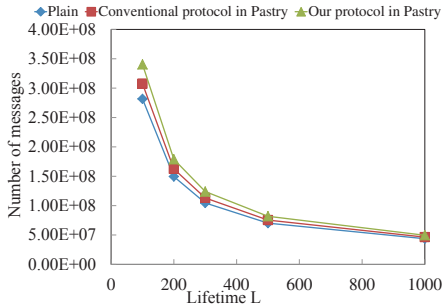

Fig. 12. The number of messages for all nodes; L = 1000.

V. CONCLUSIONS

This paper proposed a novel lightweight protocol that repairs broken pointers in structured overlay networks. Since our protocol preserves an invariancy between pointers and backpointers, all the backpointers are quickly notified compared to conventional cooperative keep-alive protocols even if churn rate is high. Unlike conventional cooperative keep-alive protocols which require each overlay node to maintain $O(\log^2 N)$ backpointers, our protocol needs only $(\log N)$ backpointers per node. These advantages of our protocol were demonstrated by comprehensive simulations.

REFERENCES

[1] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *in Proc. ACM SIGCOMM*, pages 353–366, New York, NY, USA, 2004.

[2] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays.

[3] C. Chi and K. Koyanagi. A flexible maintenance approach for structured P2P networks under churn. In *in Proc. International Conference on Communication Technology (ICCT)*, pages 273–278, 2012.

[4] I. Dedinski, A. Hofmann, and B. Sick. Cooperative keep-alives: An efficient outage detection algorithm for P2P overlay networks. In *in Proc. IEEE International Conference on Peer-to-Peer Computing (IEEE P2P)*, pages 140–150, 2007.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *in Proc. ACM Symposium on Principles of Distributed Computing(PODC)*, pages 1–12, 1987.

[6] G. Ghinita and Y.-M. Teo. An adaptive stabilization framework for distributed hash tables. In *in Proc. International Parallel and Distributed Processing Symposium(IPDPS)*, pages 1 – 10, April 2006.

[7] M. Koubarakis, C. Tryfonopoulos, S. Idreos, and Y. Drougas. Selective information dissemination in P2P networks: Problems and solutions. *SIGMOD Rec.*, 32(3):71–76, Sept. 2003.

[8] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *in Proc. USENIX Symposium on Internet Technologies and Systems*, USITS, pages 127–140, Berkeley, CA, USA, 2003. USENIX Association.

[9] K. Mizutani, S. Matsuura, S. Doi, K. Fujikawa, and H. Sunahara. An implementation and its evaluation of a framework for managing states of nodes among structured overlay networks. In *in Proc. International Conference on Networking and Service (ICNS)*, pages 282–287. IEEE Computer Society, 2010.

[10] R. Price, P. Tiňo, and G. Theodoropoulos. Still alive: Extending keep-alive intervals in P2P overlay networks. *Mob. Netw. Appl.*, 17(3):378–394, 2012.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *in Proc. IFIP/ACM International Conference on Distributed Systems Platforms(Middleware)*, pages 329–350, London, UK, 2001. Springer-Verlag.

[12] K. C. W. So and E. G. Sirer. Latency and bandwidth-minimizing failure detectors. *SIGOPS Operation System Review*, 41(3):89–99, 2007.

[13] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[14] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *in Proc. ACM SIGCOMM Conference on Internet Measurement(IMC)*, pages 189–202, 2006.

[15] S. Zhuang, D. Geels, I. Stoica, and R. H. Katz. On failure detection algorithms in overlay networks. In *in Proc. IEEE Conference on Computer Communications (INFOCOM)*, pages 2112–2123, 2005.