Sensors & Transducers, Vol. 178, Issue 9, September 2014, pp. 99-105

Sensors & Transducers

A Novel CAN Tree Coordinate Routing in Content-Addressable Network

[1] Zhongtao Li, [2] Shuai Zhao, [3] Chuan Ge, [4] Shichao Gao
[1] University Duisburg-Essen, Universität Duisburg-Essen Fakultät für Ingenieurwissenschaften Fachgebiet Verteilte Systeme Duisburg, 47048, Germany
[2] University Duisburg-Essen, Germany,
[3] Shandong University, China,
[4] China Mobile Communications Group Terminal Company Limited Shandong branch, China
[1] Tel.: 0049-17638215891
[1] E-mail: li.zhongtao@hotmail.com

Received: 22 May 2014 /Accepted: 29 August 2014 /Published: 30 September 2014

Abstract: In this paper, we propose a novel approach to improve coordination routing while minimizing the maintenance overhead during nodes churn. It bases on "CAN Tree Routing for Content-Addressable Network" [1] which is a solution for peer-to-peer routing. We concentrated on coordinate routing in this paper. The key idea of our approach is a recursion process to calculate target zone code and search in CAN tree [1]. Because the hops are via long links in CAN, it enhances routing flexibility and robustness against failures. Nodes automatically adapt routing table to cope with network change. The routing complexity is $O(\log n)$, which is much better than a uniform greedy routing, while each node maintains two long links in average. *Copyright © 2014 IFSA Publishing, S. L.*

Keywords: Coordinate routing, P2P routing, CAN routing CAN Tree, CAN, CANS.

1. Introduction

The structured approach of P2P architectures are based on analogous designs, while their search and management strategies differ. Ring-based approaches such as Pastry [3], and Chord [4] all use similar search algorithms such as binary ordered B*-tree. Content Addressable Networks (CAN) bases on the geometric space [2, 5]. The original routing of CAN has the lowest efficiency among the aforementioned structured Peer-to-Peer systems [8, 10-11]. Routing hops of Chord is $O(\log n)$ in average for a Chord circle with n participating nodes. In Pastry with n nodes, the destination is reached in $\log_{2^b}(n)$ hops. CAN can forward messages using only immediate-links. Hence, greedy routing [2] is not very efficient, particularly in large scale dynamic CAN; and CAN routing complexity is $O(d \cdot n^{1/d})$ in a d-dimensional key space [1].

In "CAN Tree Routing for Content-Addressable Network" [1], we have proposed an approach to improve peer-to-peer routing. However, we need to forward a message to a point in space without target peer information. It's coordinate routing. In this paper, we proposed a novel approach to enhance coordination routing. The routing complexity is $O(\log n)$.

2. Zone Code and CAN Tree

Instead of greedy routing, we route in a tree network. The key idea is to establish a P2P tree

(CAN tree) [6] via long links. Each node of CAN is a node of the CAN tree. Since each node only connects with its parent and child nodes in the tree network, it needs more information to choose the routing target node. For this purpose, we introduce the zone code.

We have proposed the zone code in "Using Zone Code to Manage a Content-Addressable Network for Distributed Simulations" [7]0. It's a binary string. A zone code records the splitting history of its corresponding zone. We can obtain the zone code (Fig. 1. (a) by traversing the partition tree (Fig. 1. (b)). The partition tree is a binary tree and records the reassignment process. In order to obtain a zone code, we perform a traversal from root to leaf in the partition tree. It's analogous to Hoffman code [12]. Going left is a 0, going right is a 1. A zone code is only completed when a leaf node is reached [7, 13]. Fig. 1. (b) illustrates how to establish zone codes via the partition tree. Combining all our insights, we deduced the following observation.

Fact 1: The zone code is a prefix code.

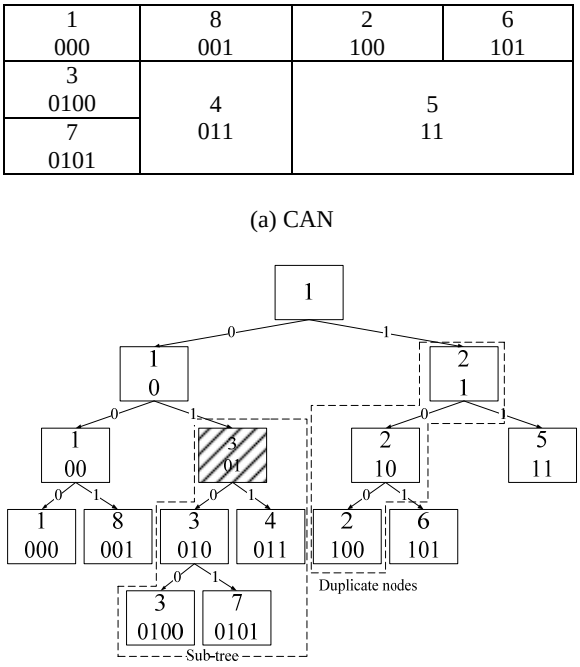


Fig. 1. CAN and partition tree.

In practice, we don't need the partition tree to generate a zone code. When a node *p* shares its half zone to a new node *c*, node *c* copies *p*'s zone code. And then node *p* and *c* append "0" and "1" respectively. Let δ^p denote the zone code of node *p*. Then, after node *c* joining, the new zone code of node *p* is $(\delta^p,0)$ and the zone code of *c* is $(\delta^p,1)$. Hence, zone code grows simultaneous with zone splitting. The more splits, the longer zone code becomes [7]. Combining all our insights, we deduced the following observation.

Fact 2: In partition tree, the zone code of node *p* is the prefix of zone codes of all nodes in the sub-tree rooted at node *p*.

For example, the node marked with shade in Fig. 1(b) as zone code (0,1). Thus, the zone codes of nodes in the sub-tree have a common prefix (0,1).

By Fact 2, we can route in the partition tree. However, the internal nodes in the partition tree do no longer exist, but were split at some previous time. The children of a node are the two nodes into which their parent node was split. Thus, we cannot establish this partition tree via long links in practice. We need the CAN tree to realize the long links.

CAN tree is a variation of the partition tree. Both of them are representations of the zone splitting process. There are some duplicate nodes that have the same name but different zone codes in the partition tree (Fig. 1(b)). If we merge duplicate nodes into one node that is parent of duplicate nodes' child nodes, it becomes CAN tree (Fig. 2). CAN tree is not a binary tree, but each node exists in CAN tree. Thus, we can implement high efficient routing in CAN tree.

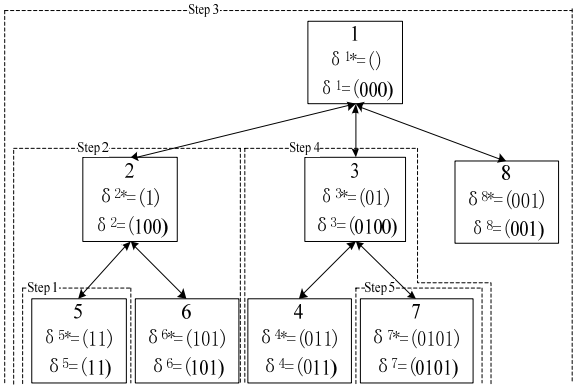


Fig. 2. CAN tree.

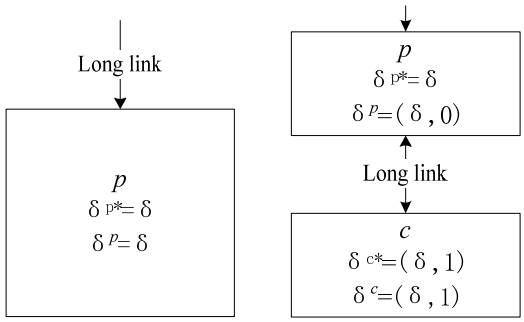
We build the CAN tree via parent-child long links. When a new node *c* forwarded JOIN request and node *p* shares half its zone with node *c*, node *c* becomes the child of node *p*. All "parent-child" relations constitute a distributed CAN tree. In order to route, each node must store its original zone code δ^* and current zone code δ. Therefore, when a new node *c* joins in CAN and obtains zone from node *p*, node *p* and *c* must act as follows (Fig. 3):

1. Node *p* splits its allocated zone in half, retaining half and handing the other half to node *c*.

2. Node *p* becomes parent of node *c*. Both of them augment long links to establish a "parent-child" relation in the CAN tree.

3. Node *c* copies *p*'s current zone code ($\delta^p = \delta$). And then, node *p* and *c* append "0" and "1" respectively, i.e. new $\delta^p = (\delta,0)$ and $\delta^c = (\delta,1)$.

4. Node *c* sets $\delta^{c*} = (\delta,1)$, Node *p* is not a new node, hence δ^{p*} does not change.



(a) Before joining (b) After joining

Fig. 3. New peer c joins CAN.

Let δ^{p*} denote the original zone code of node *p*. δ^{p*} is the first zone code of node *p*, and δ^{p*} is constant. If node *p* shared its zone to a new node, $\delta^{p*} != \delta^{p}$. For example in Fig. 2, $\delta^{3} = (0,1,0,0)$ and $\delta^{3*} = (0,1)$. δ^{p*} is the prefix of δ^{p} . Consequently, δ^{p*} is also the prefix of zone code of children of node *p*. Combining all our insights, we deduced the following observation.

Fact 3: In CAN tree, a node *p* has the original zone code δ^{p*}. The δ^{p*} is the prefix of zone codes of nodes in the sub-tree rooted at node *p*.

For example in Fig. 2 the δ^{3*} is the prefix of zone code of all nodes in sub-tree rooted at node 3. The δ^{1*} is null, it's the prefix of any zone code of nodes in the CAN tree. Since a new node obtains its zone code via copying and extending the zone code of its parent, we deduce the following:

Fact 4: If δ^{p*} of node *p* is the prefix of the δ^{c} of node *c*, node *c* is in the sub-tree rooted at node *p*.

Let δ^{p*} denote the original zone code of current node *p* and δ^{d} is the zone code of the destination node *d*. Consequently, our routing scheme is that node *p* checks whether its δ^{p*} is prefix of δ^{d} . If it is, node *p* forwards the message to its child which shares the longest common prefix with δ^{d} . If not, node *d* is not in the sub-tree rooted at node *p* and then node *p* forwards the message to its parent node.

Fig. 2 illustrates the routing from node 5 to node 7. If the destination node is not in the sub-tree rooted at current node, we expand the searching sub-tree until it covers the destination node. Afterwards, we shrink the searching sub-tree until the current node is the destination. If the destination node is in the sub-tree rooted at current node, we only shrink the searching region. During shrinking, the destination node is always in the sub-tree rooted at the current node. Thus, the routing must eventually terminate successfully.

3. Routing Mechanism

3.1. Routing Table

The routing table consists of the short links toward the neighbors and the long links toward the parent and child nodes in the CAN tree, and the original zone code δ^{*} (Fig. 4). In this section, we propose the detail of how to establish and maintain the routing table. The routing procedure is addressed in the next section.

CAN maintains short links by exchanging heartbeat messages between immediate neighbors. For d-dimensional CAN, a node maintains *O(d)* neighbors in average. This is analogous to original CAN.

Long links are a part of CAN tree. It's central to our scheme. They are established during new nodes joining. When a new node joins in CAN via Bootstrapping [14-15], an existing node splits its zone into two sub-zones, retaining one and handing the other to the new node [2]. This is same with original CAN. However, the two nodes are parent-child relationship in CAN tree. We establish long links between them, i.e. they augment a long link set in its routing table respectively. They are distant neighbors. The entry of routing table is consisted of distant neighbor information, e.g. node ID, IP address, and zone code δ (Fig. 4).

We will discuss the system responsiveness during network churn in other paper.

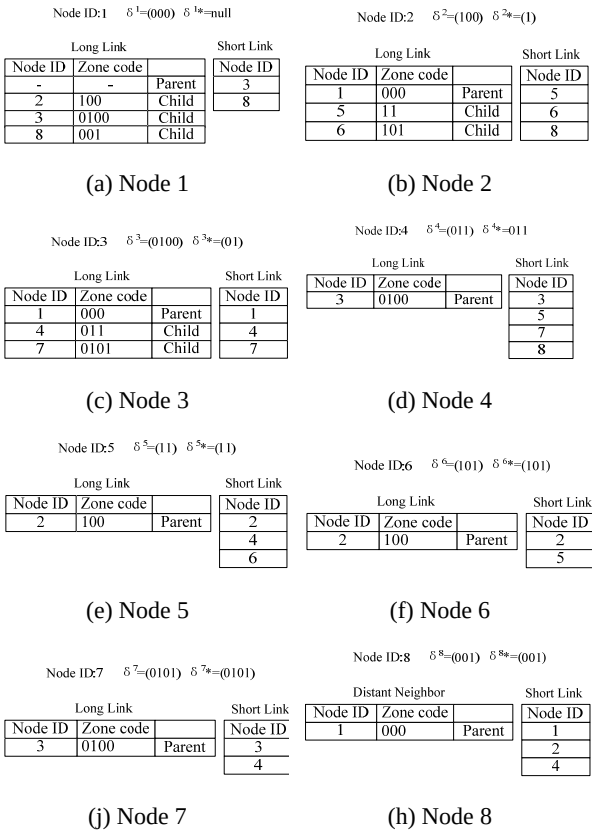


Fig. 4. Routing tables.

3.2. Get Zone Point Set Via Zone Code

By definition all zones with the same zone code length have the same size. The zone code of node p ($\delta^p = (c_1^p, c_2^p, c_3^p \cdots)$) is divided into d parts. The sub-set of the zone code $\delta_i^p = (c_{j_1}^p, c_{j_2}^p, c_{j_3}^p \cdots)$ ($j_n \bmod d = i$) records the splitting process along the i[th] axis.

Given that the zones are halved along one dimension during split, this implies that their sizes are also proportional to the inverse cof powers of 2. $|\delta_i^p|$ is the length of δ_i^p, and the proportion of p's width to space's width on the i[th] dimension is $\frac{1}{2^{|\delta_i^p|}}$. Let $(\delta_i^p)_{10}$ denote the decimal representation for δ_i^p and width w_i denote CAN's key space width on the i[th] dimension. Then $\frac{(\delta_i^p)_{10} \cdot w_i}{2^{|\delta_i^p|}}$ is p's low boundary on the i[th] dimension, and then $\frac{((\delta_i^p)_{10}+1) \cdot w_i}{2^{|\delta_i^p|}}$ is p's upper boundary on the i[th] dimension (Fig. 5).

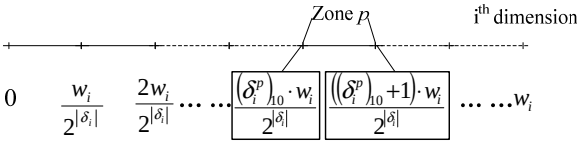


Fig. 5. Zone boundaries on one dimension.

For example, 2-dimensional CAN has the width w and height h. The zone code of p is divided into the partial zone codes δ_x^p and δ_y^p which record the x-axis and y-axis splitting process respectively. The zones are defined as a set of points $Z_{x,y}^p$:

$$Z_{x,y}^p = \left\{ (x,y) \middle| (\delta_x^p)_{10} \cdot \frac{w}{2^{|\delta_x^p|}} \le x < ((\delta_x^p)_{10}+1) \cdot \frac{w}{2^{|\delta_x^p|}} \quad (\delta_y^p)_{10} \cdot \frac{h}{2^{|\delta_y^p|}} \le y < ((\delta_y^p)_{10}+1) \cdot \frac{h}{2^{|\delta_y^p|}} \right\}$$

Equation 1 Zone point set in 2-dimensional key space

Therefore, if node 6 has zone code $\delta^6 = (1,0,1)$ in CAN (w=800 and h=600 shown in Fig. 6), it follows:

$$\delta^6 = \underset{x\ y\ x}{101} \Rightarrow \begin{cases} \delta_x^6 = 11 \\ \delta_y^6 = 0 \end{cases} \Rightarrow \begin{cases} (\delta_x^6)_{10} = 3 \\ (\delta_y^6)_{10} = 0 \end{cases} \text{ and } \begin{cases} |\delta_x^6| = 2 \\ |\delta_y^6| = 1 \end{cases}$$

$$Z_{x,y}^6 = \left\{ (x,y) \middle| (\delta_x^6)_{10} \cdot \frac{w}{2^{|\delta_x^6|}} \le x < ((\delta_x^6)_{10}+1) \cdot \frac{w}{2^{|\delta_x^6|}} \quad (\delta_y^6)_{10} \cdot \frac{h}{2^{|\delta_y^6|}} \le y < ((\delta_y^6)_{10}+1) \cdot \frac{h}{2^{|\delta_y^6|}} \right\}$$

$$Z_{x,y}^6 = \left\{ (x,y) \middle| 3 \cdot \frac{800}{2^2} \le x < (3+1) \cdot \frac{800}{2^2} \quad 0 \cdot \frac{600}{2^1} \le y < (0+1) \cdot \frac{600}{2^1} \right\}$$

$$Z_{x,y}^6 = \{ (x,y) | 600 \le x < 800 \quad 0 \le y < 300 \}$$

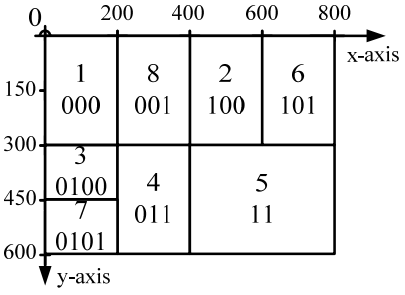


Fig. 6. CAN (width: 800 and height: 600).

3.3. Routing to a Point via Routing Table

The current node *c* sends a message to a point *p* in the key space, and we assume the destination is node *e* whose zone covers point *p*. In CAN tree coordinate routing, we also need the zone code δ^e of node *e*. However, the current node *c* does not have any information about node *e*. We can calculate the zone code via reversing the aforementioned derivation in Section 3.2, and then, forward the message to the next node. Routing is as follows:

1. Calculate δ^e: We calculate δ^e via reversing the aforementioned derivation in Section 3.2, e.g. in 2-dimensional key space, δ^e is deduced from Equation 1 . However, Equation 1 depends on the length $|\delta^e|$ of zone code of node *e*, which is an unknown factor. We assume that node *c* and *e* have the same length zone codes, i.e. $|\delta^e| = |\delta^c|$. Consequently, δ^e can be deduced from Equation 1**.**

2. Choose next node: The current node *c* checks its routing table whether its δ^{c*} is the prefix of δ^e . If it is, it forwards the message to its child node that shares the longest common prefix with δ^e . If not, it forwards the message to its parent node.

For example, node 5 in Fig. 6 has $\delta^{5*}=(1,1)$ and $\delta^{5}=(1,1)$, and forwards a message to point (100, 500). The routing procedure is as follows:

1. Node 5 assumes that node e is the destination. Since $\left|\delta^{5}\right|=2$, set $\left|\delta^{e}\right|=2$. Thus, calculate δ^{e} as follows:

$$Z^{e}_{x,y}=\left\{(x,y)\left|\left(\delta^{e}_{x}\right)_{10}\cdot\frac{w}{2^{|\delta^{e}_{x}|}}\le x<\left(\left(\delta^{e}_{x}\right)_{10}+1\right)\cdot\frac{w}{2^{|\delta^{e}_{x}|}}\quad\left(\delta^{e}_{y}\right)_{10}\cdot\frac{h}{2^{|\delta^{e}_{y}|}}\le y<\left(\left(\delta^{e}_{y}\right)_{10}+1\right)\cdot\frac{h}{2^{|\delta^{e}_{y}|}}\right.\right\}\quad \delta^{5}=\underset{x\ y}{11}\Rightarrow\begin{cases}\delta^{5}_{x}=1\\ \delta^{5}_{y}=1\end{cases}\Rightarrow\begin{cases}\left|\delta^{e}_{x}\right|=|\,\delta^{5}_{x}\,|=1\\ \left|\delta^{e}_{y}\right|=|\,\delta^{5}_{y}\,|=1\end{cases},$$

then

$$\Rightarrow(100,500)\in\left\{(x,y)\left|\left(\delta^{e}_{x}\right)_{10}\cdot\frac{800}{2^{1}}\le x<\left(\left(\delta^{e}_{x}\right)_{10}+1\right)\cdot\frac{800}{2^{1}}\quad\left(\delta^{e}_{y}\right)_{10}\cdot\frac{600}{2^{1}}\le y<\left(\left(\delta^{e}_{y}\right)_{10}+1\right)\cdot\frac{600}{2^{1}}\right.\right\}$$

$$\Rightarrow\left(\delta^{e}_{x}\right)_{10}=0\quad and\quad\left(\delta^{e}_{y}\right)_{10}=1$$

$$\left|\delta^{5}\right|=2\Rightarrow\left|\delta^{e}\right|=2$$

$$\Rightarrow\begin{cases}\delta^{e}_{x}=0\\ \delta^{e}_{y}=1\end{cases}$$

$$\Rightarrow\delta^{e}=(0,1)$$

Since $\delta^{5*}=(1,1)$ is not the prefix of $\delta^{e}=(0,1)$, it forwards the message to its parent node 2.

2. Node 2 calculates and obtains $\delta^{e}=(0,1,0)$ dependent on $\left|\delta^{2}\right|=3$ (same as 1st step). Since $\delta^{2*}=(1)$ is not the prefix of $\delta^{e}=(0,1,0)$, it forwards the message to its parent node 1.

3. Node 1 calculates and obtains $\delta^{e}=(0,1,0)$ dependent on $\left|\delta^{1}\right|=3$. Since it is root node, it forwards the message to child node 3 whose $\delta^{3}=(0,1,0,0)$ shares the longest common prefix with δ^{e}.

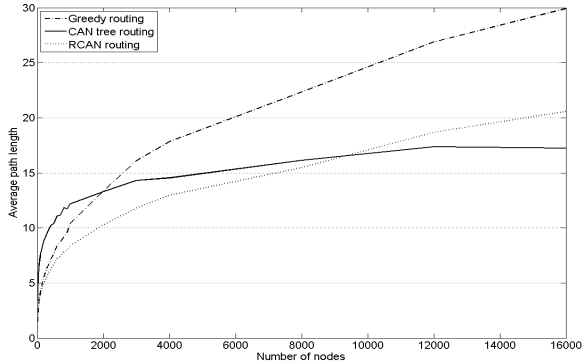
4. Node 3 calculates and obtains $\delta^{e}=(0,1,0,1)$ dependent on $\left|\delta^{3}\right|=4$. Since $\delta^{3*}=(0,1)$ is the prefix of δ^{e}, it forwards the message to child node 7 which is the destination. Thus, routing is finished.

4. Evaluation

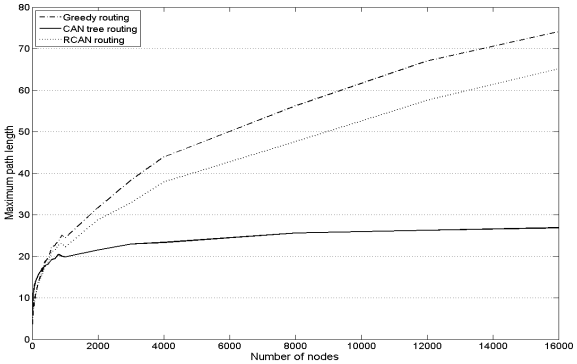
Our solution did not re-design CAN Tree routing, but extended it. By this novel approach, we could forward a message to an unbeknown point. The routing procedure always converges, since each step forwards the message to a node which shares a longer prefix than the last step, each step moves closer to the destination.

The complexity depends on the tree structure. In order to demonstrate the effectiveness of our design in terms of routing performance, we have implemented a CAN tree routing scheme in C# and conducted a set of experiments via distinct schemes on networks with up to 16000 nodes. We run CAN tree routing against original CAN greedy and RCAN routing to offer comparative measurements. These measures include essentially: path length to cope with different network size, path length distribution, and number of long links per node.

Fig. 7(a) and Fig. 7(b) plot respectively the average and the maximum path length with respect to network size. The path length is measured by the number of hops traversed during each lookup request. Fig. 7 illustrates that both the average and maximum path length in CAN tree routing are better than in other routing, and both of them are perfectly asymptotic to the logarithm of nods. Path length of greedy routing (Fig. 7) increases much faster.



(a) Average path length



(b) Maximum path length

Fig. 7. Path length with increasing network size.

Fig. 8 illustrates the path lengths distribution of routing in CAN with 16000 nodes. The path length distribution of greedy routing is much better than in other routing.

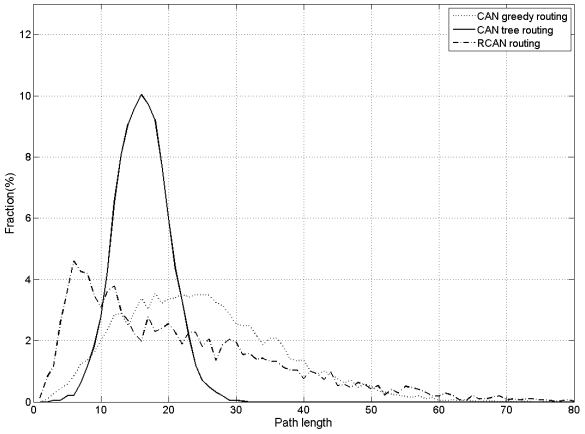


Fig. 8. Path length distribution.

Except first node, every new node needs two long links to join in CAN tree. Each parent node needs one long link pointing to its child; and child nodes need one long link pointing to its parent. Hence, number of long links is $(n-1)\times 2$, and average long links can be calculated as:

$$L_{average} = \frac{(n-1)\times 2}{n}$$

$$\lim_{n \to \infty} L_{average} = \lim_{n \to \infty} \frac{(n-1)\times 2}{n} = 2$$

Thus, each node maintains two long links in average.

CAN tree coordinate routing has same complexity, path length distribution and average long links with CAN Tree peer-to-peer routing [1].

5. Conclusions

CAN tree coordinate routing is distinguished from peer-to-peer routing. It's a novel routing scheme. Current node calculates target zone code and compare with its zone code to choice next hop. Therefore, we could forward a message to a point, while we do not have any information about the point's owner.

CAN with CAN tree routing is a completely decentralized system. CAN tree infrastructure gracefully adapts itself to cope with any changes in the network. As a pure peer-to-peer system, nodes assume equal responsibility. System maintains nodes' routing states with minimizing cost even in the presence of high rate of churn. The critical contribution is to equip each node with long links that extremely enhance routing efficiency. The amount of long links per node is independent of the network size. The system can scale by several orders of magnitude without loss of efficiency.

Our routing scheme has more links than original CAN, which causes a tiny overhead to maintain long links. It's proved that the number of long links per node is 2 in average. However, it also shows the small extension leads to significant improvements on routing performance.

Our ongoing work includes the investigation of advanced mechanisms for load balancing [9] to improve the system responsiveness during network churn.

References

[1]. Zhongtao Li, Torben Weis, CAN Tree Routing for Content-Addressable Network, *Sensors & Transducers*, Vol. 162, Issue 1, January 2014, pp. 124-130.

[2]. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, A Scalable Content Addressable Network, in *Proceedings of the ACM SIGCOMM*, 2001.

[3]. A. Rowstron, P. 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)*, November 2001, pp. 329-350.

[4]. I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications, in *Proceedings of the ACM SIGCOMM Conference*, 2001, pp. 149–160.

[5]. Ralf Steinmetz, Klaus Wehrle, Peer-to-Peer Systems and Applications, *Springer*, 2005.

[6]. Zhongtao Li, Torben Weis, Content-Addressable Network for Distributed Simulations, in *Proceedings of the International Conference on Communications, Mobility and Computing (CMC'12)*, May 2012.

[7]. Zhongtao Li, Torben Weis, Using Zone Code to Manage a Content-Addressable Network for Distributed Simulations, in *Proceedings of the IEEE 14th International Conference on Communication Technology (ICCT'12)*, Nov. 2012.

[8]. Xu Z., Zhang Z., Building low-maintenance expressways for P2P systems, *Technical Report HPL-2002-41 41, HP Laboratories*, Palo Alto, 2002.

[9]. Sahin O. D., Agrawal D., Abbadi A. E., Techniques for efficient routing and load balancing in content-addressable networks, in *Proceedings of the 5th IEEE Intl. Conference on Peer-to-Peer Computing (P2P 2005)*, Los Alamitos, Washington, DC, USA, 2005, pp. 67–74.

[10]. Sun X., SCAN: a small-world structured P2P overlay for multi-dimensional queries, in *Proceedings of the 16th Intl. Conf. on World Wide Web (WWW'07), ACM*, New York, 2007, pp. 1191–1192.

[11]. Boukhelef D., Kitagawa H., Multi-ring Infrastructure for Content Addressable Networks, in *Proceedings of the 16th International Conference on Cooperative Information Systems (CoopIS'08), Lecture Notes in Computer Science*, Vol. 5331, 2008, pp. 193-211.

[12]. David A. Huffman, A Method for the Construction of Minimum-Redundancy Codes, in *Proceedings of the IRE*, 1952.

[13]. Zhongtao Li, Shuai Zhao, A. J. Han Vinck, Ru Jia, Efficient Codes for Writing Equal-bit Information in a WOM Twice, *Journal of Multimedia*, Vol. 9, Issue 4, Apr. 2014, pp. 477-482.

[14]. M. Knoll, A. Wacker, G. Schiele, T. Weis, Decentralized bootstrapping in pervasive applications, in *Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07)*, 2007, pp. 589-592.

[15]. M. Knoll, A. Wacker, G. Schiele, T. Weis, Bootstrapping in Peer-to-Peer Systems, in *Proceedings of the 14th International Conference on Parallel and Distributed Systems (ICPADS'08)*, Melbourne, Victoria, Australia, 8-10 December, 2008.