

CAN Tree Routing for Content-Addressable Network

Zhongtao LI, Torben WEIS

University Duisburg-Essen, Universität Duisburg-Essen Fakultät für Ingenieurwissenschaften
Fachgebiet Verteilte Systeme Duisburg, 47048, Germany
Tel.: 0049-17638215891
E-mail: li.zhongtao@hotmail.com

Received: 5 November 2013 / Accepted: 9 January 2014 / Published: 31 January 2014

Abstract: We propose a novel topology to improve the routing performance of Content-Addressable Network overlays while minimizing the maintenance overhead during nodes churn. The key idea of our approach is to establish a P2P tree structure (CAN tree) by means of equipping each node with a few long links towards some distant nodes. The long links enhance 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: CAN Tree, CAN, CANS, P2P routing, CAN routing.

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 [2], and Chord [3] all use similar search algorithms such as binary ordered B*-tree. Content Addressable Networks (CAN) bases on the geometric space [1, 4].

The original routing of CAN has the lowest efficiency among the aforementioned structured Peer-to-Peer systems. 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 $\log_2(n)$ hops. CAN can forward messages using only immediate-links. Hence, greedy routing [1] 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.

We know the average routing path length i.e., the number of nodes traversed during routing, from CAN

simulator [1]. Fig. 1 illustrates the average routing path length in each case for dimensions 2 to 5. Because increasing the number of dimensions implies that each node has more neighbors, and each node has more potential next hop nodes [1]. The results indicate that the routing efficiency is low especially for low dimensions. In order to reduce the routing latency in CAN, the original CAN proposes to increase the number of immediate neighbors per node via enhancing dimensions. Unfortunately, our Content Addressable Network for Simulation (CANS) [5] is utilized to simulate “city traffic” and MMVE. “City traffic” and MMVE require a 2-dimensional or 3-dimensional space. Hence we need an efficient routing solution in low dimensions.

Long links have been extensively utilized by many other P2P protocols to improve routing performance such as Chord [3], Pastry [2]. Moreover, we have eCAN [7], LDP [8], SCAN [9], and RCAN [10] (see 2. Related Work), which have also adopted long links for the same purpose of improving routing functionality but in different ways [10]. They are built on top of the conventional CAN overlay.

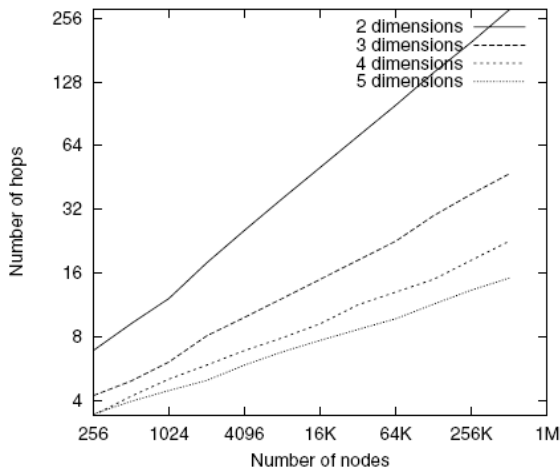


Fig. 1. Effect of dimensions on path length [1].

Our scheme is also based on long links. However, we concentrate on a novel routing via establishing search tree infrastructure in CAN in order to improve its routing performance and enhance fault-tolerance. Meanwhile, both of long links and node churn maintenance overhead should be minimized.

2. Related Work

Content-Addressable Network (CAN) is proposed in [1]. It is based on geometric design, and achieves $O(d \cdot n^{1/d})$ routing performance with $O(d)$ routing state per node. Recently some research works concentrated on improving lookup efficiency in CAN. Their common essence is via additional long links to reduce the routing latency and enhance fault tolerance.

In “Building Low-maintenance Expressways for P2P Systems” [7], they proposed “eCAN”, which is designed for efficient routing via expressways in CAN. eCAN routing performance can achieve $O(\log n)$. Expressways are established by snapshot. Cartesian key space is divided into topology areas of different spans called “expressway zones”. Expressway zones are hierarchical. Low level expressway zone is resident of high level’s expressway zone. Each node owns a CAN zone and is also a resident of the expressway zones. Hence, any message needs to be propagated to upper level. The construction of eCAN may have direct impact on the routing flexibility, scalability and fault-tolerance [7]. eCAN is closely related to our work. Because it builds expressway according to snapshot, eCAN can not immediately adjust itself according to network changing. Subsequently, snapshot generates extra overhead except long links.

“Long distance pointers” (LDP) [8] introduced distinct method to establish long links via randomly choosing nodes in CAN. LDP number is a fixed parameter, which does not change with the nodes number. Hence, it does not guarantee that LDP

evenly distributes across the key space. The authors proposed also an improved scheme “sub-space pointers” (SSP). After splitting the Cartesian key space into virtual sub-spaces, each node establishes long links to random nodes in each virtual sub-space.

In RCAN [10], each node establishes some sets of long links, each of which is established along one dimension. Long link forwards to distances inverse of power of 2 from the originating node [10]. It achieves $O(\log n)$ maintenance overhead after node churn. Moreover, the average number of long links per node is $d \log_d^n$ in a d-dimensional RCAN.

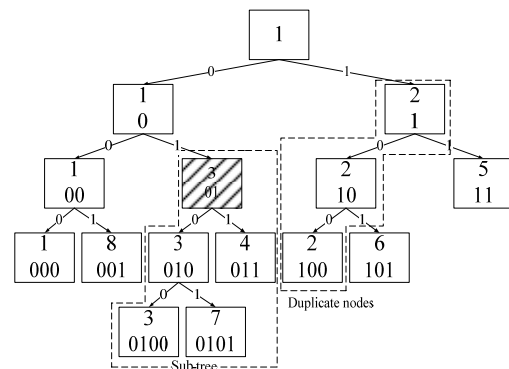
3. Zone-code and CAN Tree

Instead of greedy routing, we route in a tree network. The key idea is to establish a P2P tree (CAN tree) [5] 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” [6]. It’s a binary string. A zone-code records the splitting history of its corresponding zone. We can obtain the zone-code Fig. 2(a) by traversing the partition tree Fig. 2(b).

1 000	8 001	2 100	6 101
3 0100	4 011	5 11	
7 0101			

(a) CAN



(b) Partition tree

Fig. 2. CAN and partition tree.

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 [11]. Going left is a 0, going right is a 1. A zone-code is only completed when a leaf node is reached [6]. Fig. 2(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.

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 [6]. 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. 2(b) has 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. 2(b). If we merge duplicate nodes into one node that is parent of duplicate nodes' child nodes, it becomes CAN tree (Fig. 3). CAN tree is not a binary tree, but each node exists in CAN tree. Thus, we can implement high efficient routing in CAN tree.

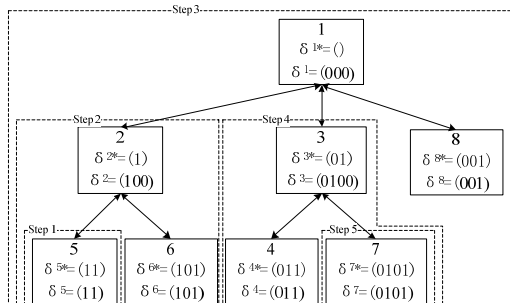


Fig. 3. CAN tree.

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. 4):

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 ($\delta^c = \delta$). 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.

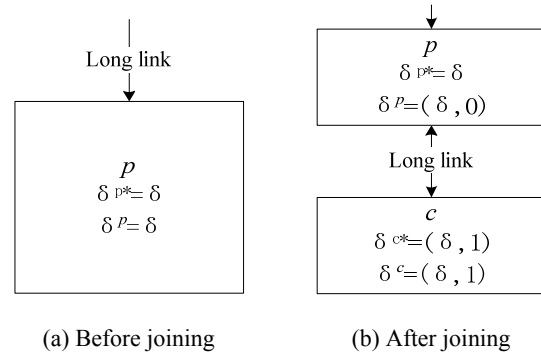


Fig. 4. New peer c joins CAN.

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, $\delta^{p*} \neq \delta^p$. For example in Fig. 3, $\delta^3 = (0,1,0,0)$ 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. 3, 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. 3 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.

4. 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. 5).

Node ID:1 $\delta^1=(000)$ $\delta^{1*}=\text{null}$			
Long Link			Short Link
Node ID	Zone code		Node ID
-	-	Parent	3
2	100	Child	8
3	0100	Child	
8	001	Child	

Fig. 5 (a). Routing tables: node 1.

Node ID:2 $\delta^2=(100)$ $\delta^{2*}=(1)$			
Long Link			Short Link
Node ID	Zone code		Node ID
1	000	Parent	5
5	11	Child	6
6	101	Child	8

Fig. 5 (b). Routing tables: node 2.

Node ID:3 $\delta^3=(0100)$ $\delta^{3*}=(01)$			
Long Link			Short Link
Node ID	Zone code		Node ID
1	000	Parent	1
4	011	Child	4
7	0101	Child	7

Fig. 5 (c). Routing tables: node 3.

Node ID:4 $\delta^4=(011)$ $\delta^{4*}=011$			
Long Link			Short Link
Node ID	Zone code		Node ID
3	0100	Parent	3
			5
			7
			8

Fig. 5 (d). Routing tables: node 4.

Node ID:5 $\delta^5=(11)$ $\delta^{5*}=(11)$			
Long Link			Short Link
Node ID	Zone code		Node ID
2	100	Parent	2
			4
			6

Fig. 5 (e). Routing tables: node 5.

Node ID:6 $\delta^6=(101)$ $\delta^{6*}=(101)$			
Long Link			Short Link
Node ID	Zone code		Node ID
2	100	Parent	2
			5

Fig. 5 (f). Routing tables: node 6.

Node ID:7 $\delta^7=(0101)$ $\delta^{7*}=(0101)$			
Long Link			Short Link
Node ID	Zone code		Node ID
3	0100	Parent	3
			4

Fig. 5 (j). Routing tables: node 7.

Node ID:8 $\delta^8=(001)$ $\delta^{8*}=(001)$			
Distant Neighbor			Short Link
Node ID	Zone code		Node ID
1	000	Parent	1
			2
			4

Fig. 5 (h). Routing tables: node 8.

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 [12][13], an existing node splits its zone into two sub-zones, retaining one and handing the other to the new node [1]. 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. 5).

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

5. Routing Mechanism

If a node has the zone-code δ^d of a destination node, it can exactly choose the next hop via its routing table. Fig. 6 illustrates the flow diagram of routing. If current node c is not the destination node, it checks whether its original zone-code δ^{c*} is a prefix of δ^d . If it is, it forwards the message to the distant node that shares the longest common prefix zone-code with δ^d . If not, it forwards the message to its parent node. Since the root node has no parent in CAN tree, it always forwards the message to its child, which shares the longest prefix with δ^d . As expressed Fact 4 in section 3, the routing must eventually terminate successfully.

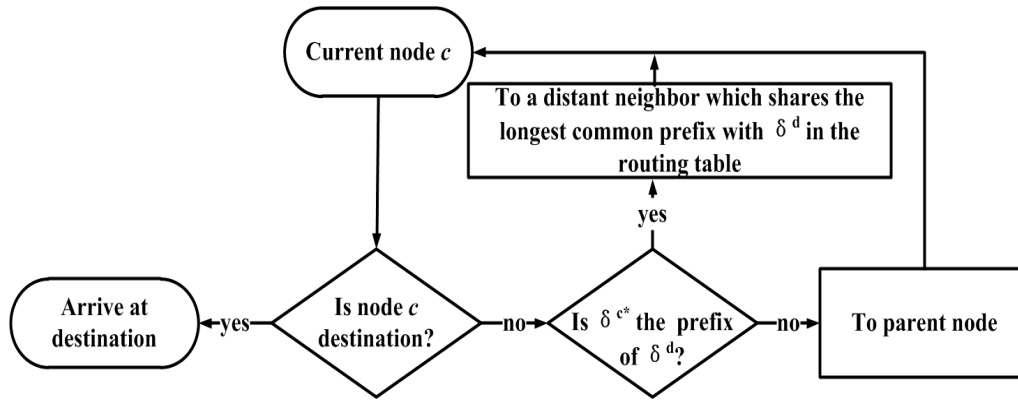


Fig. 6. Flow diagram.

For example, node 5 ($\delta^5 = (1,1)$ and $\delta^{5*} = (1,1)$) in Fig. 3 forwards a message to node 7 ($\delta^7 = (0,1,0,1)$). The routing table is in Fig. 5. The routing is as follows:

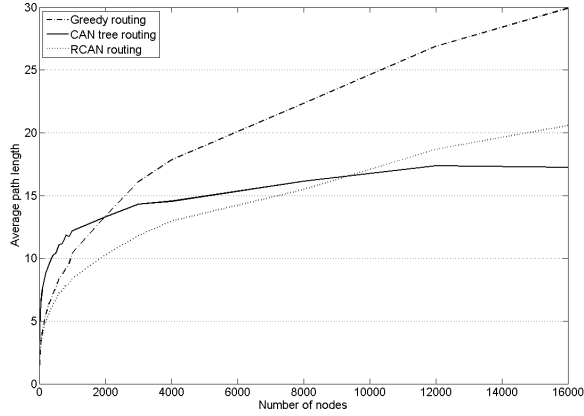
1. Node 5 is not the destination. Since $\delta^{5*} = (1,1)$ is not the prefix of $\delta^7 = (0,1,0,1)$, node 5 forwards the message to its parent node 2.
2. Node 2 is not the destination. Since $\delta^{2*} = (1)$ is not the prefix of $\delta^7 = (0,1,0,1)$, node 2 forwards the message to its parent node 1.
3. Node 1 is not the destination. Since it is root node, it forwards the message to the child node 3 whose $\delta^3 = (0,1,0,0)$ shares the longest common prefix zone-code with $\delta^7 = (0,1,0,1)$.
4. Node 3 is not the destination. Since $\delta^{3*} = (0,1)$ is the prefix of the $\delta^7 = (0,1,0,1)$, node 3 forwards the message to child node 7 which is the destination. Thus, routing is finished.

6. Evaluation

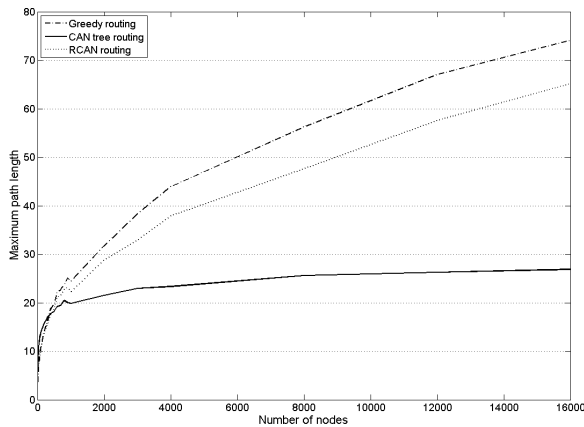
Our solution did not re-design CAN routing, but extended it. Via long links, routing is optimized several features simultaneously: small routing path, more routing flexibility and fault-tolerance. The routing procedure always converges, since each step forwards the message to a node which shares a longer prefix than the last step, each step moves closer to the destination.

CAN tree routing is based on a tree. Hence 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 nodes. 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.

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:

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

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

Thus, each node maintains two long links in average.

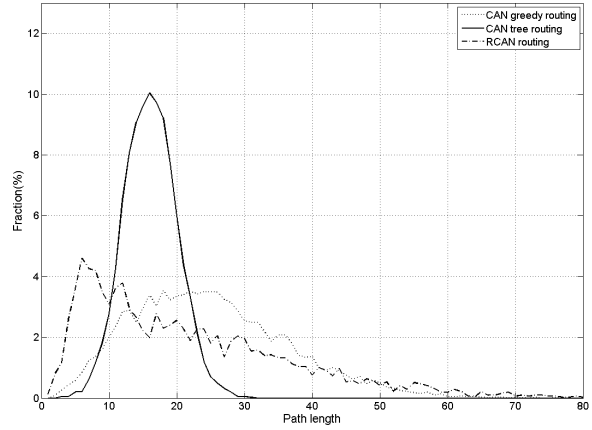


Fig. 8. Path length distribution.

7. Conclusion and Future Works

We proposed in this paper CAN tree routing, a novel routing scheme based on CAN tree to overcome the weakness of routing in CAN. 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.


We are also investigating the issue of load balancing in the present scheme. Our ongoing work includes the investigation of advanced mechanisms for load balancing to improve the system responsiveness during network churn.

References

- [1]. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, A scalable content addressable network, in *Proceedings of the ACM International Conference on Applications*,

- Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM' 01)*, San Diego, California, USA, 27-31 August 2001, pp. 161-172.
- [2]. A. Rowstron and 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)*, Springer, Heidelberg, Germany, November 2001, pp. 329-350.
 - [3]. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in *Proceedings of the ACM Conference (SIGCOMM '01)*, 2001, pp. 149-160.
 - [4]. Ralf Steinmetz and Klaus Wehrle, Peer-to-peer systems and applications, *Springer*, 2005.
 - [5]. Zhongtao Li, Torben Weis, Content-addressable network for distributed simulations, in *Proceedings of the International Conference on Communications, Mobility, and Computing (CMC'2012)*, May 2012.
 - [6]. 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' 2012)*, November 2012.
 - [7]. Z. Xu, Z. Zhang, Building low-maintenance expressways for P2P systems, Technical Report HPL-2002-41 41, *HP Laboratories*, Palo Alto, 2002.
 - [8]. O. D. Sahin, D. Agrawal, A. E. Abbadi, Techniques for efficient routing and load balancing in content-addressable networks, in *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P'2005)*, Washington, DC, USA, 2005, pp. 67-74.
 - [9]. X. Sun, SCAN: a small-world structured P2P overlay for multi-dimensional queries, in *Proceedings of the 16th International Conference on World Wide Web (WWW' 07)*, ACM, New York, 2007, pp. 1191-1192.
 - [10]. D. Boukhelef, and H. Kitagawa, Multi-ring infrastructure for content addressable networks, in *Proceedings of the 16th International Conference on Cooperative Information Systems (CoopIS '08)*, Monterrey, Mexico, November 12-14, 2008, *Lecture Notes in Computer Science*, Vol. 5331, Springer Berlin, Heidelberg, 2008, pp. 193-211.
 - [11]. David A. Huffman, A method for the construction of minimum-redundancy codes, in *Proceedings of the IRE*, 1952.
 - [12]. M. Knoll, A. Wacker, G. Schiele, and T. Weis, Decentralized bootstrapping in pervasive applications, in *Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW' 07)*, 2007, pp. 589-592.
 - [13]. M. Knoll, A. Wacker, G. Schiele, and T. Weis, Bootstrapping in peer-to-peer systems, in *Proceedings of the 14th International Conference on Parallel and Distributed Systems (ICPADS'2008)*, Melbourne, Victoria, Australia, December 8-10, 2008.

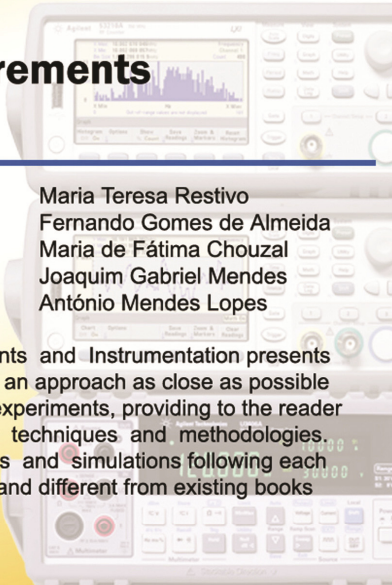
2014 Copyright ©, International Frequency Sensor Association (IFSA). All rights reserved.
(<http://www.sensorsportal.com>)




Handbook of Laboratory Measurements and Instrumentation

Maria Teresa Restivo
 Fernando Gomes de Almeida
 Maria de Fátima Chouzal
 Joaquim Gabriel Mendes
 António Mendes Lopes

The Handbook of Laboratory Measurements and Instrumentation presents experimental and laboratory activities with an approach as close as possible to reality, even offering remote access to experiments, providing to the reader an excellent tool for learning laboratory techniques and methodologies. Book includes dozens videos, animations and simulations following each of chapters. It makes the title very valued and different from existing books on measurements and instrumentation.





International Frequency Sensor Association Publishing

Order online:
http://www.sensorsportal.com/HTML/BOOKSTORE/Handbook_of_Measurements.htm