Development of a Prototype System of RCAN P2P System

Djelloul Boukhelef [1]

Advisor: Pr. Hiroyuki Kitagawa [1,2]

Abstract. RCAN is a novel multi-ring content addressable overlay system. RCAN was proposed in the aim of improving the routing performance of CAN overlays while minimizing the maintenance overhead during nodes churn in large networks. The key idea of RCAN is to equip each node with a few long-links towards some distant nodes. Long-links are clockwise directed and wrap around to form small rings along each dimension. The number of rings and their sizes self-adjust as nodes join and leave the system. RCAN is a pure P2P design, where all nodes assume the same responsibility. Unlike some existing P2P overlays, RCAN is self-organizing and does not assume any a-priori fixed limits for the network size or the routing state per node. Each node in RCAN auto-adapts its routing state to cope with network changes. Our aims is to propose a new P2P overlay protocol with several optimized features simultaneously: small routing path, small routing state, low maintenance cost during nodes churn, and more routing flexibility and fault-tolerance.

In this report, we will give a *zoom-in* description of the design and implementation of a prototype system of RCAN, and highlight its layered and modular architectures. A simulation-based approach was used, in which an initial simulation model of RCAN system is iteratively transformed into a real network-based system. We will show our current realization and discuss some future extensions of our RCAN protocol as well as its prototype system. The goal of our current work is to develop a prototype system of RCAN protocol that can be easily deployed in a real computer network (or even at an Internet-scale in the future). Another ultimate objective is to propose a general framework that supports developer in simulating, developing, and testing different and heterogeneous P2P protocols.

1. INTRODUCTION

Peer-to-peer (P2P) is a new emerging paradigm for organizing large-scale self-organizing distributed systems. P2P systems are fundamentally different than traditional client-server systems, in the sense that they do not employ any central authority nor assume any global knowledge. P2P systems are dynamic by nature where nodes can join and leave the network freely. Participating nodes act simultaneously as clients and servers and exchange information and services directly with each other. Each node keeps contacts with other nodes in the system.

P2P systems can be classified into two categories: unstructured and structured. In unstructured systems, nodes are unaware of the resources that their neighboring nodes maintain. Lookup is typically performed by flooding the query message to all participating nodes (e.g. Gnutella). Structured P2P implement a *Distributed Hash Table* (DHT) functionality to deterministically map resources into nodes. Each node maintains information about resources its neighbors provides. DHT protocols have become an important class of P2P systems due to their scalability, routing efficiency and search completeness. Representative protocols include: CAN [6], Chord [7], etc.

筑波大学大学院システム情報工学研究科
1. Graduate School of Systems and Information Engineering, University of Tsukuba
2. Center for Computational Sciences, University of Tsukuba

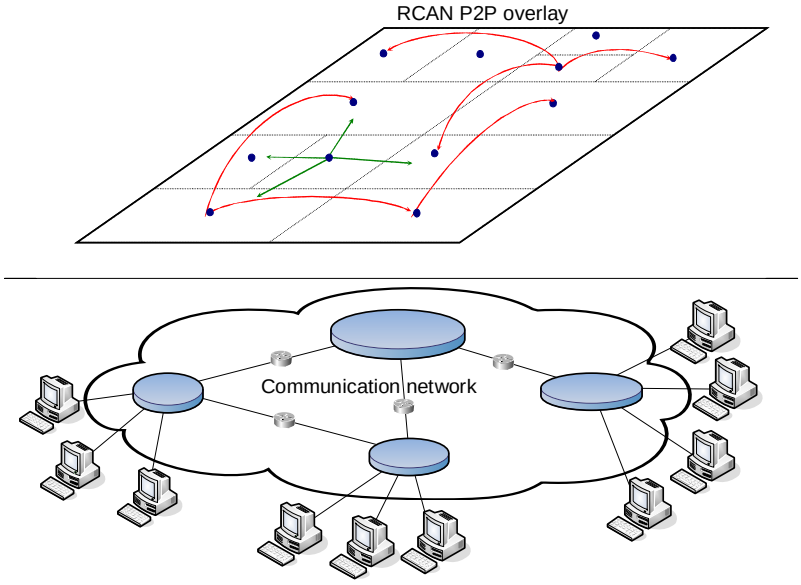


Figure 1. The distributed architecture of RCAN system. Each blue point is a node. The red and green lines represent long and short links, respectively.

2. MULTI-RING CONTENT ADDRESSABLE NETWORK (RCAN)

RCAN is a novel self-scaling multi-ring content addressable network. RCAN proposes a new P2P protocol with a new topological and routing infrastructure. The basic substrate of RCAN is a conventional CAN overlay, where a node knows only about its immediate neighborhood. Currently, the greedy routing using only neighboring nodes is not efficient and is more vulnerable to network failures [2,3]. The key idea of RCAN is to consider equipping each node with few *long-range links* (*long links*, for short) towards some distant nodes (called *distant neighbors*). Long links are established in such a way that the routing path is short while the maintenance overhead for building and updating routing tables when nodes join or leave the system is very low. A node selects distant neighbors situated at distances inverse of powers of 2 on the coordinate space. The set of long links in each node is partitioned into small sub-sets, each of which is established along one dimension. Long links are clockwise-directed and wrap around the key space.

Unlike some close related P2P topologies, RCAN is a self-scaling system where no upper limits for the number of links per node or the network size are imposed. Each node maintains a routing state that automatically self-scales when the network size changes.

A self-organizing P2P overlay has the ability to spontaneously adapt its self to continuous changes without the need of an external or central authority. These changes include changes in node membership due to churn, and imbalance in data and routing loads. In this perspective, our aim is to propose a system where each node autonomously adapts its local routing state to cope with changes in the network size.

In a dynamic P2P system, nodes join and leave the system frequently, which may partially impair the overlay predefined structure and reduce system performance. Therefore low-overhead stabilization process is highly needed to restore the system structure and to keep the system performance at an acceptable level. Following this idea, we proposed an efficient approach to maintain the routing information during nodes churn with low communication overhead. In RCAN, at most O(log n) messages are exchanged in order to build the routing table of a newly joining node, O(log n) messages are also needed to update all the invalid entries in the other O(log n) affected routing tables.

In a self-organizing system, nodes should cooperate to ensure fair load balancing among them, and provide more routing flexibility and fault-tolerance. RCAN provides also a cost-free yet efficient

mechanism to cope with moderate load imbalance. However, due to space limitation we will not go further on this aspect.

The set of long links from different nodes yields multiple independent rings along each dimension. The rings are of small size, and their maintenance is very easy. The number of rings and theirs sizes self-adjust as nodes join or leave the network. In a uniformly partitioned key space, a node is member of only one ring per dimension, i.e. the ring that intersects with its own region.

2.1. System Model

In this section we will describe the Multi-ring Content Addressable Network, RCAN. Its aim is to provide a novel approach for establishing long links on top of a CAN-like overlay in order to improve its routing performance and enhance its fault-tolerance capabilities while minimizing the maintenance overhead during nodes churn.

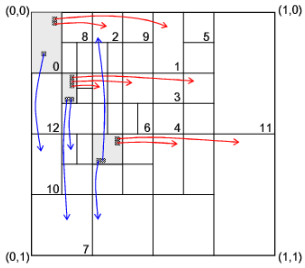


Figure 2. Design principle of RCAN.

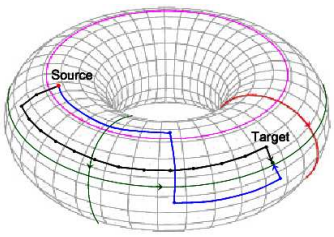


Figure 3. 2D toroidal model of RCAN

RCAN operates on a virtual *d*-dimensional Cartesian key space (Figure 2). The coordinate space warps around each dimension. A data item (*object*) is a (*key* , *value*) pair, where *key* is a *d*-dimensional vector and *value* is non-key information associated with the object (string, etc.).

The key space is divided into non-overlapped hyper-rectangular regions. Their sizes could be changed dynamically as nodes join (*split*) or leave the system (*merge*). Each region *r* is given a globally unique identifier (*r.id*) generated by applying a hash function to some data points from the region *r* itself.

Regions are assigned to nodes using a *one-to-one* mapping. Each node *p* is responsible for the data items that fall in its region *r*. The logical address of *p* is the same as its region's *id* (*p.id* = *r.id*). Given that nodes and regions are objectively related and their *id*s are fixed, we will simplify our notation and refer to both the node (resp. region) and its identity as *p* (resp. *r*) (without ~.*id*).

2.2. Join operation

When a new node *q* joins the system, it contacts an existing node *p* to share the load with. Node *p* splits its region into two equal parts. One of which is handed to *q*. Logically, on the virtual splitting tree *p* is a parent of *q*. But on the flat key space their region are also sibling. A region is split along one dimension each time, in a cyclic way (dimension 0,1,…*d*-1; 0,1, etc). As we mentioned in the introduction, the join procedure is slightly modified such that a new node join at a position in the key space that preserve a good load distribution among the existing nodes.

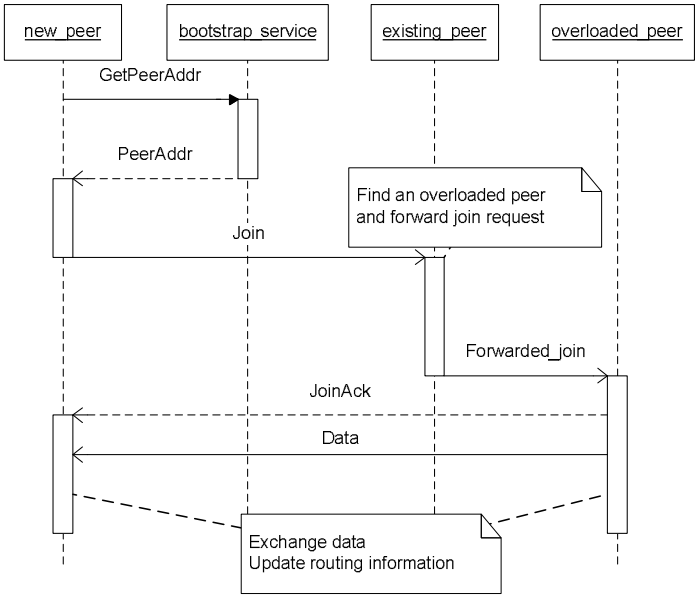


Figure 4. Simplified sequence diagram of a *join operation*

2.3. Leave operation

When a node *q* is about to leave the system, it contacts one of its neighbors, let's say *p*, hands its regions and data items to *p* and then leaves the system. If the node *p* is the latest sibling of *q*, it will extend its responsibility by merging its region with the region of *q*. Otherwise; *p* handles temporarily the tow regions, and will look for the latest sibling of *q*'s region in order to finalize the merge process.

RCAN is a decentralized self-organizing content addressable network. Multiple splits and merges can be conducted independently at the same time. As a result, regions with different extents may coexist in the key space. Each node maintains its local region and keeps contact with its neighboring nodes. Only little information is exchanged between neighbors to perform split or merge operations. No information is spread out to other distant nodes or committed to a central server.

3. ROUTING INFRASTRUCTURE

For routing purpose, each participating node maintains certain routing information about its neighboring nodes.

In RCAN, the routing state of each node consists of two types of links: *short links* towards adjacent neighbors and *long links* towards distant nodes. When we say that a node *p* has a link towards a node *q*, this means that a direct communication channel is established between the two nodes, and through which *p* can send messages to *q*.

3.1. Short Links

Short links are contact information (network address, region's extent, etc.) about adjacent nodes. A node has an average of O(*d*) immediate neighbors, independent of the network size [6]. A node stores its short links in a local routing table, called **`CTable`**. The maintenance of **`CTable`** is similar to CAN, for example, by exchanging heartbeat messages between neighboring nodes.

3.2. Long Links

Long links are at the core of our work. In this section, we show how to establish and maintain long links and how to build RCAN's multi-ring infrastructure.

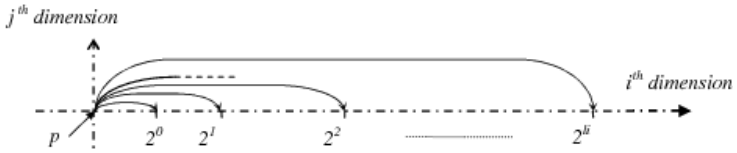


Figure 5. Long links along the i-th dimension.

The routing state of a node is augmented with small number of unidirectional links towards some distant nodes in the system. Assume that a node n keeps k_i long links along the i-th dimension (Figure 4). Denote them $l_i = \{l_i^0,\ldots,l_i^{ki-1}\}$, where l_i^j is the distance between n and the node pointed by the j-th link. l_i is an ordered set, that is, $l_0 < l_2 \ldots < l_{ki-1}$. The value of $l_i^j = 2^j \times w_i$, where w_i is the size (width) of the region of n along the i-th dimension. k_i indicates also how many time the region of n had been split along the i-th dimension. Each set l_i of long links built along the i-th dimension is stored in a separate routing table, called **`LTable`**$_i$.

From the above definition we can see that the number of long links originating from each node is proportional to the size of its region. A node, hence, can adapt dynamically the number of long links by establishing additional links when its region is split (node join), or dropping extra links when nodes leave the system. Experiments show that the total number of long links per node $k = \Sigma k_i$ is tightly bounded by O(log n).

3.3. Multi-ring Topology

Long links are parallel to data axes and wrap around the key space. Long links originating from nodes whose origins are situated on the same line form a ring along each dimension. Multiple small rings are hence formed. In a regular grid, a node is member of one ring along each dimension, that is, the ring intersecting with its region.

The number of rings and the number of nodes per ring self-scales when the network size changes. In a key space where its regions may have different sizes, a node may temporarily become a member of more than one ring along each dimension because its region is large and it intersects with several rings.

3.4. Routing Algorithm

In RCAN we consider a *hop-by-hop* greedy routing approach. A node uses only local routing information to decide the next step. To forward a message, the current node consults its routing tables and determines the node that is strictly closer to the target among its *immediate* and *distant* neighbors, according to a well defined metric[1], and forwards the message through that node. If there are many neighbors at the same *expected* distance to the target, the node may choose one at random.

The routing process in RCAN consists of solving the routing path along one ring at each step. During the routing process, rings can be used in an arbitrary order (Figure 3). A good feature of RCAN multi-ring topology is that there are many paths with *almost* the same expected distance between any pair of nodes. This property enables more flexibility in the selection of the path to route a message to its target, and gives RCAN routing mechanism more robustness against node and links failures, by offering each node more choices to select the best neighbor through which it route a message to its target.

[1] RCAN uses clockwise Manhattan distance as metric function. Other metrics may also be used.

4. PROTOTYPE OF RCAN

In this section, we present a *zoom-in* description of RCAN's design and implementation, highlighting its layered and modular architectures.

Figure 5 shows the overall architecture of the RCAN's prototype system that we have developed. It basically shows the differentiation of the three major components: *management module*, *overlay layer*, and *communication interface*.

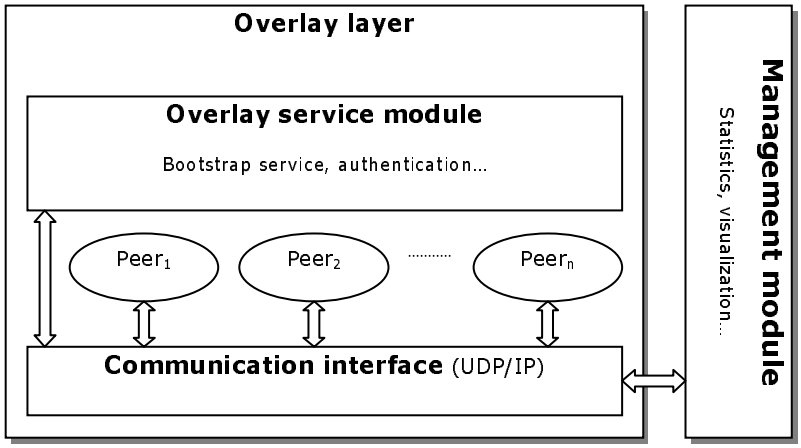


Figure 6. Layered architecture of RCAN prototype

4.1. Overlay Layer

Overlay layer is the core component of our prototype system. This layer provides basic and essential functionalities of any distributed data/services management system, such as: Publish/subscribe, Lookup, querying, ... RCAN protocol is designed to be deployed in a real network environment. For that reason, the *overlay layer* has been decomposed into tow main sub-modules: o*verlay core layer*, and o*verlay management module*.

A. Overlay core layer

The *Overlay core layer* is the running environment for *peer applications*. It provides the basic data management functionalities (lookup, insert, routing...). Basically, *overlay core layer* is a protocol-dependent component. In our current realization, RCAN, CAN, LDP, and SSP protocols are all supported.

B. Overlay management module

The *overlay management* module provides high-level overlay management interface such as bootstrapping, security... In a P2P framework, this layer may be implemented as a centralized component (our case), or distributed into several cooperating components (*agents*). Unlike the *overlay core layer*, the *overlay management module* should offer a generic interface to access and manage the underlying P2P systems, which is independent of the overlay protocol.

As consequence, different overlay protocols layers (RCAN, Chord...) can be implemented (with slight changes if necessary), and integrated under the same *overlay management framework*.

C. Connectivity middleware

The *connection middleware* is a software component installed in single-copy on each physical machine. Its main task is to ensure connectivity between peers in the *overlay core layer* (currently implemented as a pool of threads) and the *overlay management module*, by exchanging some vital information such as: network addresses (IP@ and port numbers) of the overlay management layer, administration layer; creating and destroying peers,...

Our architectural design offers more flexibility, extensibility, and robustness. Which makes our RCAN' protocol an interesting framework for implementing, and developing P2P protocols as well as experimenting some advanced topic such as the interoperability.

4.2. Management module

The *management module* provides a higher-level overlay visualization and administration interface. The *management module* is responsible for collecting information about peers and overlay application, as well as the underlying communication network. Then it compiles them to produce statistics about the overlay system (routing, links...). This information is presented to the system administrator in a suitable and user-friendly fashion.

The *management module* uses periodic probing and soft states to detect any changes in the overlay composition, the status of peers (including peer arrival, failure), query routing, and construct a *statistical* map of the overlay.

4.3. Communication Interface

The *communication interface* provides abstraction from the underlying communication network peculiarities. It provides a generic and uniform interface (classes and routines) for efficient communication. Basically, the communication interface is deployed on top of the IP protocol stack, and offers a *message passing* interfaces for connection/connectionless oriented modes with both synchronous and asynchronous models.

A light version of windows *IO completion protocol* model is also implemented by this layer. Note that all the inter-application and intra-application interactions use the message-passing communication interface. This design choice makes our architecture more generic and easily extensible to other communication models, and intimately easily deployable on a real network. These needs show the importance of the communication interface, which requires more performance and robustness.

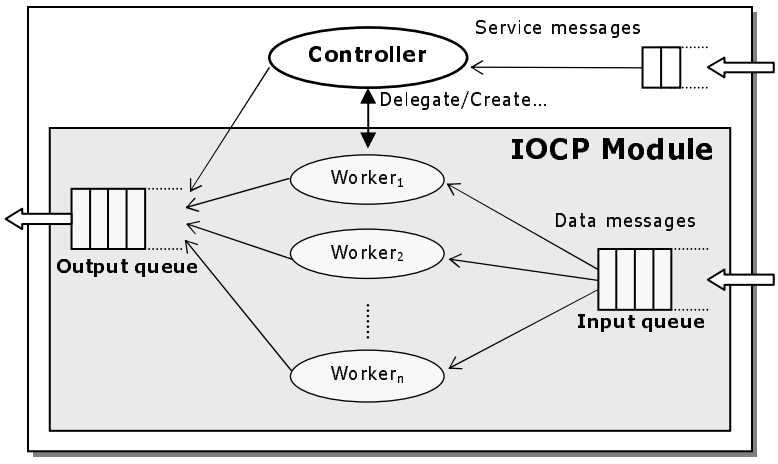


Figure 7. Application Internal architecture (Control and communication layers)

5. INTERNAL ARCHITECTURE

The internal architecture of the three applications, that is, *peer*, *overlay*, and *manager* is shown is represented on figure 7. This architecture is based on windows IOCP model, where one main *Controller* thread is responsible for managing the application, and the creation/destruction of worker threads…A synchronization mechanism is also provided to ensure the integrity of the application data.

For the implementation we used both windows worker (*callback*) and interface thread (`WinThread`) models. The synchronization between thread uses mainly *locks*, and the *critical section* mechanisms.

6. IMPLEMENTATION

We followed an object-oriented approach to design and implement RCAN prototype system. Figure 8 depicts the class diagram of the main classes of our prototype.

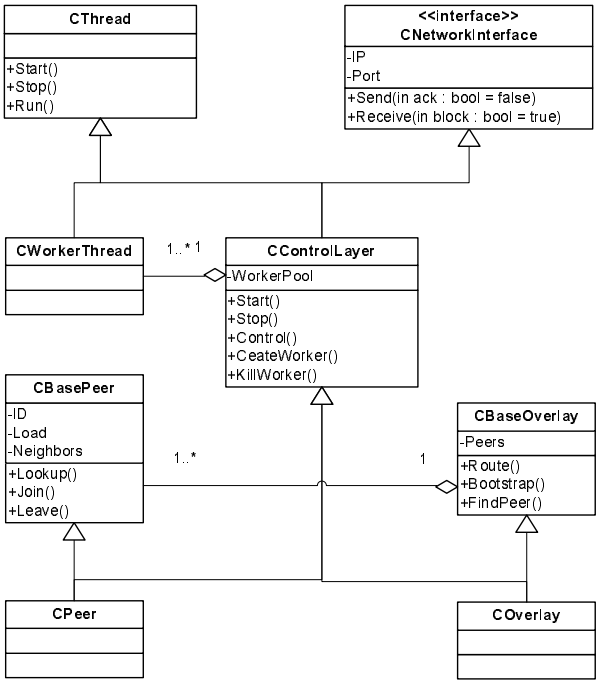


Figure 8. Class diagrams

`CNetworkInterface` provide a network communication interface for both synchronous and asynchronous modes. In our implementation we used user datagram protocol (UDP) for message passing between the different applications.

`CControlLayer` provides a general framework to implement network-based application based on windows IOCP model. As we mentioned in the previous section, `CPeer` and `COverlay` are both inherited from `CControlLayer`.

For visualization/simulation purposes we have split `CPeer` and `COverlay` classes into two levels: `CBasicPeer` and `CBasicOverlay` offer basic centralized functionalities such storage, routing…. The lower level classes implement advanced network functionalities such as network communication, concurrency…

7. USER INTERFACE

We have implemented a multi-threaded version of RCAN protocols using C++. In this new version all the communications between peers, the overlay module and the manager uses UDP/IP. Our aim is to make the protocol easily deployable on a real network. In what follows we show some part of the user interface of RCAN prototype.

7.1. Kernel Application (Simulator)

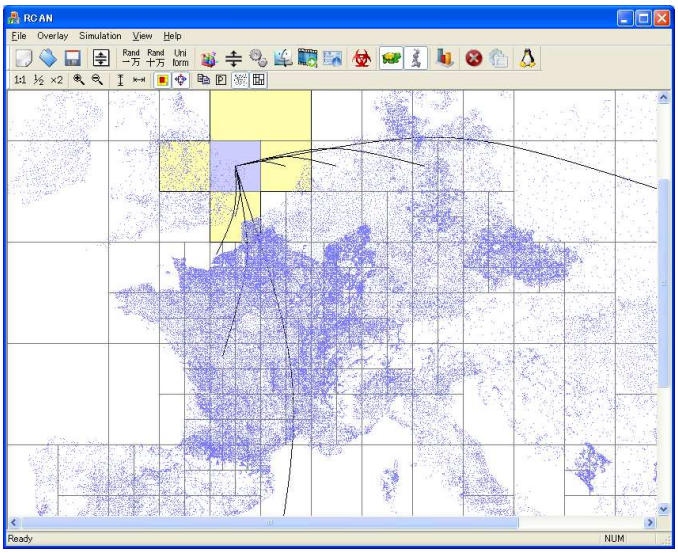
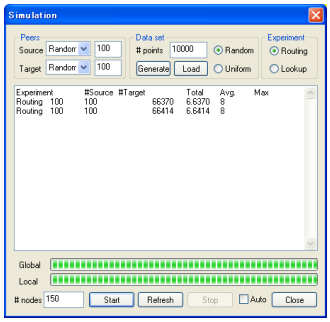


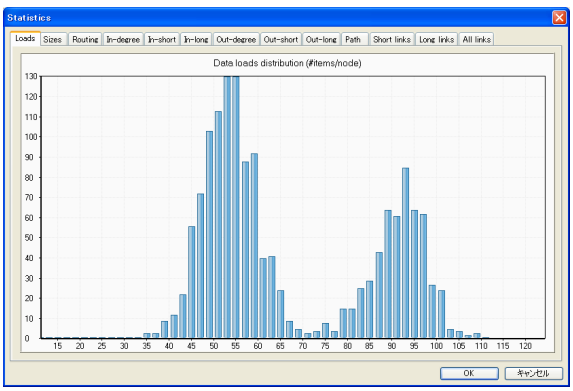
Figure 9. Main window (simulator)

(a)



(b)

Figure 10. Create an overlay (a) and perform simulation (b)



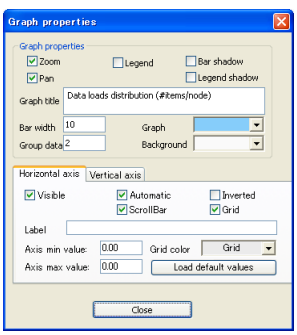


Figure 11. Statistics visualization

“Simulator” represents the kernel of RCAN protocol. It implements thread-based peers, and overlay modules, and provides simple and rich commands to create overlays (figure 10.a), performs simulations (figure 10.b) and retrieves a statistics about the experiment (figure 11).

7.2. Manager application

This is a separate application of the RCAN simulator. It mainly provides a remote interface to manipulate the simulator parameters. In a real network application, this application may play the role of remote agent for the connection/creation/management of peer applications.

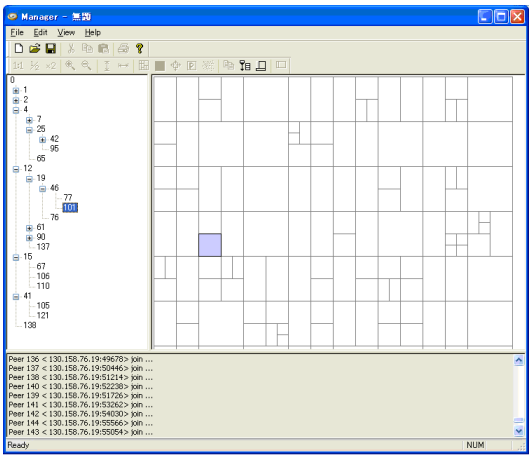


Figure 12. Screenshot of Manager

8. EVALUATION

In order to demonstrate the effectiveness of our design, we have implemented an RCAN protocol in C++ and conducted a set of experiments using different data patterns on networks with up to thousands of nodes. We have analyzed the behavior of RCAN under different uniform and skewed data patterns: Uniform data set is generated using C random function. Real-world data set consists of the coordinates of about 680000 main towns in Europe.

We evaluate RCAN in term of routing performance, maintenance cost, load balancing, as well as some advanced bootstrapping mechanisms. We also used this prototype to prove the robustness of our RCAN routing and load balancing mechanisms against some network attacks. In what follows we present some of the experimental results. For more details and interpretations please refer to the basic papers about RCAN

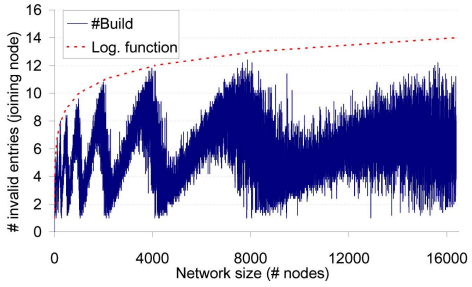


Figure 13. Maintenance cost of join operation (# messages)

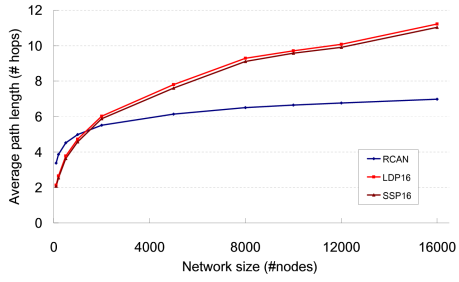


Figure 14. Path length as function of the network size

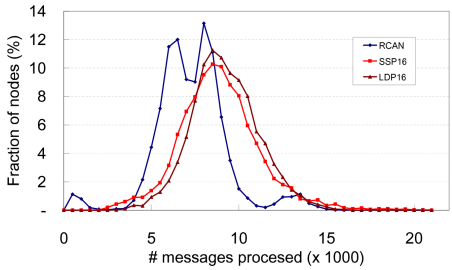


Figure 15.Routing loads distribution (#processed messages/node)

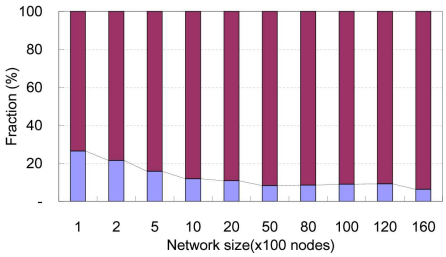


Figure 16. Links usage patterns

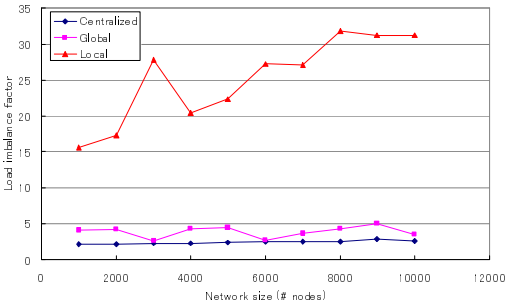


Figure 17. Load imbalance factor

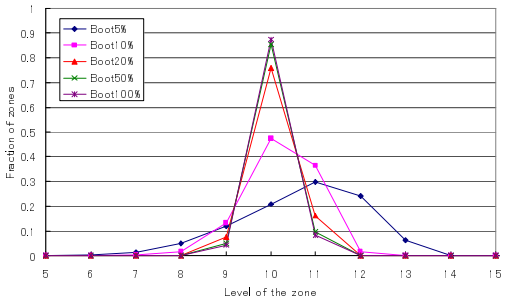


Figure 18. Robustness under adversarial attacks

9. CONCLUSION

We proposed an implementation of RCAN, a novel self-organizing topological structure to overcome the weakness of greedy routing in CAN overlay. The key idea is to equip each node with additional long-range links. RCAN is a pure P2P system where nodes assume the same responsibility. Nodes can join and leave the system autonomously with the minimum maintenance overhead. Each node maintains a routing state that self-scales logarithmically with the network size. RCAN's multi-ring infrastructure gracefully adapts itself to cope with any changes in the network membership.

RCAN protocol is simple and efficient and shows that even a small extension can lead to significant improvements on different aspects. We must emphasize that RCAN is by no mean the first P2P system that uses long links for this purpose. It is not also the first to achieve logarithmic routing performance using logarithmic routing state. However, to the best of our knowledge, RCAN is the first CAN-like overlay that provides a completely decentralized mechanism that provides a self-scaling routing state. The number of long links per node is proved to be O(log n). RCAN is also the first to achieve O(log n) maintenance overhead during nodes churn. This makes RCAN optimal on this aspect in comparison with other existing methods that incur a $O(\log^2 n)$ maintenance overhead.

RCAN constitute an interesting substrate to deploy fair load balancing strategies and to improve node join protocol. RCAN increases network connectivity which makes the routing process more flexible and fault-resilient. We proposed in this work a simulation-based implementation of a prototype system of RCAN protocol. Besides of the performance and scalability requirements, our aim is to provide an architectural design with some key features such as: flexibility, extensibility, robustness...

10. REFERENCES

[1] Boukhelef, D. and Kitagawa, H. RCAN: *A Multi-ring Content Addressable Network*. iDB Forum 2008, Fukushima, Japan. September 21-23, 2008. IPSJ SIG Technical Reports 2008-DBS-146. Vol. 2008, No. 88. pp. 91-96.

[2] Boukhelef, D. and Kitagawa, H. *Multi-ring Infrastructure for Content Addressable Networks*. Proceedings of the 16th International Conference on Cooperative Information Systems (CoopIS'08), Monterrey, Mexico, November 12 - 14, 2008. pp. 193-211. . Lecture Notes in Computer Science, vol. 5331. Springer Berlin, Heidelberg, 193-211.

[3] Boukhelef, D. and Kitagawa, H. *Efficient Routing in Multi-ring Content Addressable Network.* DBSJ Journal, Vol.7, No.3. December 2008.

[4] Boukhelef, D. and Kitagawa, H. *Dynamic Load Balancing in RCAN Content Addressable Network.* Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC'09). SKKU, Suwon, Korea. January 15-16, 2009.

[5] Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: *A survey and comparison of peer-to-peer overlay network schemes*. IEEE Communications Surveys and Tutorials 7(2) (2005) 72–93

[6] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: *A scalable content addressable network.* In: Proc. of ACM-SIGCOMM. (2001) 161–172

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

[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: Proc. of the 5th IEEE Intl. Conf. on Peer-to-Peer Computing (P2P'05), Washington, DC, USA, IEEE Computer Society (2005) 67–74

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

[11] Surana, S., Godfrey, B., Lakshminarayanan, K., Karp, R., and Stoica, I. (2006). *Load balancing in dynamic structured peer-to-peer systems*. Performance Evaluation. 63, 3 (2006), 217-240.

[12] Kenthapadi, K. and Manku, G. S. (2005). *Decentralized algorithms using both local and random probes for P2P load balancing*. In Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (Las Vegas, Nevada, USA, July 18 - 20, 2005). SPAA '05. 135-144.

[13] Manku, G. S.(2004). *Balanced binary trees for ID management and load balance in distributed hash tables*. In Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing (St. John's, Newfoundland, Canada, July 25 - 28, 2004). PODC '04. 197-205.

[14] Hughes, C. and Hughes, T. (2003) *Parallel and Distributed Programming Using C++*. Prentice Hall Professional Technical Reference. Addison-Wesley Professional (September 2003).

[15] Loo, A, (2006) Peer-to-Peer Computing: *Building Supercomputers with Web Technologies* (Computer Communications and Networks). Springer; 1 edition (December, 2006).

[16] Gomaa, H. (2000) *Designing Concurrent, Distributed, and Real-Time Applications with UML* (Addison-Wesley Object Technology Series). Addison-Wesley Professional (September, 2000).