Routing in Content Addressable Networks: Algorithms and Performance

Alexandru Popescu[†‡], David Erman[†], Markus Fiedler[†] and Demetres Kouvatsos[‡]

[†] *Dept. of Telecommunication Systems*
School of Engineering
Blekinge Institute of Technology
371 79 Karlskrona, Sweden

[‡] *Dept. of Computing*
School of Informatics
University of Bradford
Bradford, West Yorkshire BD7 1DP, United Kingdom

***Abstract*—**

Over the last years, virtual networking has evolved towards becoming a dominant platform for the deployment of new services and applications such as large-scale data sharing, multimedia services, application-level multicast communication. A consequence of this evolution is that features like robust routing, efficient search, scalability, decentralization, fault tolerance, trust and authentication have become of paramount importance.

Since network virtualization basically represents the process of combining resources of different types, two distinct aspects stand out. The first aspect is virtualization of hardware on network entities, while the second being in essence a virtualized network such as an overlay network. In this context, overlay networks act as enablers by providing the addressing and routing substrates needed to implement a virtualized network structure.

In this paper, we focus on the second aspect of network virtualization and consider the specific case of structured overlay networks with a particular focus on Content Addressable Networks (CAN). An investigation of the existing approaches for structured P2P overlay networks is provided, where we point out their advantages and drawbacks. The essentials of CAN architecture are presented and based on that, we report on the implementation of a CAN simulator. Our initial goal is to use the simulator for investigating the performance of the CAN with focus on the routing algorithm. Preliminary results obtained in our experiments are reported as well. The results indicate that greedy routing algorithms perform better than non-greedy algorithms.

I. Introduction

Over the last years, the Internet has been the place of tremendous technology success, which now enable larger access to information, provide new ways of communication among people and organizations and fundamentally change our way to work and to learn. At the same time, this success is now revealing the fundamental limitations of the current Internet design, with negative consequences on future networking applications and services. While many efforts have been done on the development of new services and applications, little has been done on deploying these facilities in the Internet at large. The Internet is simply requested to allow the co-existence of different designs and services in the same architectural framework. This is however not the best solution. Problems like, e.g., inappropriate security and trust, inefficient mobility handling, lack of support for multi-homing and difficulties in the deployment of content-centric networking, have revealed the need for further efforts to de-ossify the Internet and to allow users take full advantage of the new applications. For instance, the well-known difficulties related to the deployment of relatively modest architectural changes like IPv6, indicate the strong need for further research and development efforts to solve these problems.

Network virtualization basically represents the process of combining resources of different types like, e.g., hardware, software, services, into a single administrative entity. The diverse types of network resources are used in this case through a logical segmentation of the physical network.

The notion of network virtualization is rather new, and no clear consensus has been reached on the actual definition of the term. However, two complementary aspects of the term can be observed in the current literature. The first aspect focuses on the virtualization of the hardware on network entities, such as routers and gateways. This type of virtualization has typically been used to provide virtual server capacity on existing hardware, using software such as VMware or XEN [3]. In the second aspect, a virtualized network is in essence an overlay network, which imposes a new networking substrate on top of already existing ones. The most obvious of this type of virtual network are the P2P networks that appeared in the late 1990's. In this paper, we focus on the latter, overlay network, aspect of network virtualization.

Network virtualization is an interesting solution for addressing the problem of Internet ossification. Multiple virtual networks may co-exist on top of a shared substrate. Virtual networks can be implemented by diverse virtual elements like, e.g., virtual routers connected by virtual links. The recent emergence of high performance processors have had as a consequence that network elements like virtualized routers can now be built with performance comparable to conventional routers at the advantage of greater flexibility. In this context, overlay networks act as enablers by providing the addressing and routing substrates needed to implement a virtualized network structure. Overlay networks, in the form of P2P networks, have become very popular over the last years due to features that make them suitable for the development or deployment of new services like overlay multicast communi-

cation, large-scale data sharing and content distribution.

P2P networks exhibit three fundamental features: self-organization, symmetric communication and distributed control [10]. The self-organized routing can be implemented in two ways: in a structured or unstructured fashion. Structured P2P networks are subject to a logical structure governing the network formation, and are ofted implemented in the form of Distributed Hash Tables (DHTs), providing both routing and addressing. One such structured overlay network is Content Addressable Networks (CANs), combining location and content in the addressing scheme. This makes it an interesting prospect for use in virtualized networks, in particular for data-intensive networks. Given this background, this paper will focus on the performance of routing in a CAN, in order to evaluate it as an enabler for routing in virtualized networks.

This paper is structured as follows. In Section II we provide a short overview of the state of the art research in P2P systems. This is followed in Section III by a description of how CAN work. Our own implementation of the CAN routing protocol along with simulation results are described in Sections IV and V respectively. Finally, in Section VI we present some brief conclusions and ideas for future work.

II. Routing in P2P Systems

P2P systems have been shown to be an efficient platform for developing scalable and robust distributed applications. Facilities like diversity of concepts, direct access among distributed computers, sharing of computer resources by direct exchange, self-organizing facilities and decentralized resource administration make them very attractive for virtual networking. Today, some of the most important issues related to P2P systems are related to naming, indexing, routing, congestion control and security [14].

There are two main classes of P2P networks with reference to the routing substrate, namely unstructured networks and structured networks [1]. Unstructured networks, also called "first generation" P2P networks, use flooding and random walks for routing. Better solutions have therefore been developed to alleviate the large overhead traffic by, e.g., using super-peers, clustering, selective forwarding or a combination thereof. Typical examples are Gnutella and Kazaa. The unstructured networks have the main advantage in their simplicity but this is associated with serious drawbacks like large routing costs and difficulties in providing scalability when handling increased rates of aggregate queries or when the system size increases.

On the other hand, structured networks, also called "second generation" P2P networks, typically use Distributed Hash Table (DHT) routing schemes to reduce the routing cost and to provide a bound on the number of hops required for localizing a target data item [2]. Such systems have important advantages like, e.g., decentralization, scalability, availability, short routing distances and fault tolerance. DHT routing is based on the concept of prefix-based routing initially introduced by Plaxton to accommodate dynamic join/departure of peers and to provide failure-recovery mechanisms [13]. In other words, structured P2P networks means that the P2P network topology is tightly controlled and data objects are placed at specific locations selected such as to obtain better query performance.

Examples of such networks are Plaxton, Pastry, Tapestry, Chord and CAN, which are employing different DHT routing algorithms [10], [17]. Graph-theoretic properties are used to determine and to improve the efficiency of such systems [9], [15], [16]. Typically, such graphs are of $\Theta(\log n)$ diameter and show $\Theta(\log n)$ degree at each node, where n is the number of peers in the system. CAN however has a different performance profile, as it chooses the keys from a d-dimensional Cartesian space, as discussed in Section III. CAN nodes have $\Theta(2d)$ neighbors and the pathlengths are of $\Theta(d/4)(\, n^{1/d})$ hops. However, in the case of logarithmic CAN, i.e., when $d = \log n$, CAN shows similar properties like the other networks, i.e., $\Theta(\log n)$ diameter and $\Theta(\log n)$ degree at each node.

A serious problem of structured systems is related to churn, with serious impact on the network performance. Other drawbacks are significantly higher overheads, the lack of support for keyword searches and complex queries. However, recent efforts towards the development of a unification platform for different DHT-systems are making structured networks more and more attractive [2], [7], [14]. Such a platform is expected to provide a API based on the KBR (Key-Based Routing) concept as described in [2], coupled with a basic DHT service model to easily deploy DHT-based applications.

Another important issue is regarding the query mechanisms used in P2P systems and ways to optimize them. A query mechanism is used to construct efficient searches from user input. There are several query systems existing today, e.g., range queries, multi-attribute queries, join queries and aggregation queries, each of them with their own advantages and drawbacks [17]. The challenge in this case is to develop new models for query optimizations for large networks in the order of thousands and tens of thousands of servers and millions of clients [8].

The fundamental criteria for developing efficient DHT routing algorithms is to provide the best tradeoff with reference to a set of measures like, e.g., routing efficiency, resilience to node failure, routing hot spots and geography-based performance [9], [15], [16]. Other measures like, e.g., minimum resource consumption, are important as well, particularly in the case of mobile ad-hoc networks. Routing efficiency refers to the best tradeoff of minimizing the routing pathlength in combination with the associated state. Related to this, we have implemented four different CAN routing algorithms and their implementation and performance is reported in our paper. Resilience to node failures demands for a rather sophisticated analysis, where diverse aspects need to be analyzed like, e.g., static resilience, resilience time and cost. Routing hot spots refers to the efficiency of diverse schemes suggested for solving this problem. Geography-based performance refers to the analysis of real-world systems as well as developing of efficient algorithms to reduce the end-to-end latency.

III. CAN Essentials

CAN is a distributed Internet-scale hash table designed to map file names to a specific network location. It is a robust, scalable, and decentralized system designed for efficiently locating data stored in a DHT [14]. The key space in a CAN is an n-dimensional Cartesian coordinate space, which wraps around the edges of each dimension, thus creating a n-dimensional torus geometry. For the experiments presented in this paper, we have used a 2-dimensional coordinate space using the x, y coordinates.

Every node in the CAN is identified by a point P in the key space. Additionally, the node is responsible for its own *zone*, which is a rectangular portion of the key space that surrounds the point P. The node has information about adjacent zones and their responsible nodes. Nodes route messages in the CAN overlay using only information about neighboring nodes and their corresponding zones. Since the CAN space is a 2-dimensional coordinate grid, this becomes a matter of routing along the shortest path which in this case is a straight line between two points. The construction of a CAN overlay consists of three steps: bootstrapping, finding a zone and joining the overlay routing. These correspond to the functions[1] BOOTSTRAP, FINDZONE and JOINROUTING outlined in Algorithm 1.

The purpose of bootstrapping is to enable the node to find the IP address of an existing CAN node. In the initial proposal of CAN [15], a particular bootstrapping procedure is not defined, but the authors suggest a solution similar to that used in YOID [5]. This means that a DNS lookup of a CAN domain will reveal the IP address of a bootstrap node (a.k.a. *rendezvouz point*). The bootstrap node is then used to obtain a set of IP addresses corresponding to active CAN nodes.

After successfully completing the bootstrap procedure, the joining node must find its own zone. To do this, the node randomly picks a node b from the list supplied by the bootstrap node, and a coordinate on the x-axis one on the y-axis and assigns the values to the point P. P then implicitly becomes the identifier of the joining node.

The new node then tries to find a zone in the CAN, which contains P. To do this, it assembles a JOIN message and asks node c to route it towards point P. Upon receiving the JOIN message, node p, which is responsible for the zone where P belongs, executes the GETZONE procedure. The result of the procedure is the division of p's zone into two equal parts, where p keeps one part and relinquishes the other, Z, to the new node. The procedure returns Z together with the set of neighbors, N, responsible for zones adjacent to Z.

When the new node has found its zone, all nodes in the system send an immediate update to their neighbors to inform them if any change has occurred in their zone. This is followed by periodic update messages where similar information is exchanged.

[1]Procedure names preceded by a variable name and a dot indicate remote function calls (e.g., n.LOOKUP means that node n executes the LOOKUP function).

Algorithm 1 CAN construction

```
1:  S ← BOOTSTRAP
2:  c ← RAND(S)
3:  P ← RAND(X,Y)
4:  Z, N ← FINDZONE(c, P)
5:  JOINROUTING(N)

6:  procedure BOOTSTRAP
7:      Contact a DNS server d
8:      b ← d.LOOKUP(CAN domain)          ▷ bootstrap node
9:      c ← b.GETCANNODES                 ▷ set CAN nodes
10:     return c
11: end procedure

12: procedure FINDZONE(c, P)
    Route JOIN message towards point P via node c
13:     Z, N ← p.GETZONE                  ▷ P is in p's zone
14:     return Z, N
15: end procedure

16: procedure JOINROUTING(N)
17:     Send soft updates to all nodes N
18: end procedure

19: procedure LOOKUP(domain)
20:     Lookup IP address ip associated with domain
21:     return ip
22: end procedure

23: procedure GETCANNODES
24:     return subset of known CAN nodes
25: end procedure

26: procedure GETZONE
27:     Give up half of own zone, Z, to calling node
28:     Collect the set N of neighbors to half-zone Z
29:     return Z, N
30: end procedure
```

If a node leaves the CAN, one of its neighbors assumes responsibility of the leaving node's zone. Zone information is then refreshed during periodic routing updates. The absence of routing messages from a neighbor is taken as an indication of a node failure, and a takeover mechanism is initiated. The purpose of the takeover mechanism is to re-assign the zone of the failed node to one of its neighbors in a consistent manner, i.e., preventing several nodes from simultaneously attempting to take over the zone [15].

IV. Implementation of CAN Routing

We have implemented the CAN construction algorithm as well as several routing algorithms. Our implementation creates a 2-dimensional $[0, 1] \times [0, 1]$ coordinate space for the CAN. The first node to join the CAN becomes the owner of the entire CAN space and it is assigned the coordinates $\{0,0\}$.

Additionally, it is selected as bootstrap node. This is just a matter of convenience and has no impact on anything else other than the bootstrapping procedure.

As previously mentioned, if the random point P is not located within the zone owned by the bootstrapping node, then a JOIN request must be routed through the CAN. Starting from the bootstrap node, the routing operation is accomplished by attempting to follow a straight line through the Cartesian space from source to destination. This is done by one of the four implemented CAN routing algorithms explained below. In a 2-dimensional coordinate space two nodes are neighbors if their coordinate spans overlap along one dimension and abut along the other dimension. In the current CAN implementation, the routing operation always starts from the bootstrap node located at the origo in the coordinate space. For each node on the path, the algorithm computes the extremities of each neighbor zone to find the one with the shortest distance to P conforming to the currently used routing algorithm. For each neighbor zone the distance to P is computed from eight different points: four of them are the zone corners and the remaining four are the middle points on each zone border. The neighbor, whose zone contains the point with the shortest distance to P, is a candidate for the next hop on the path. To avoid loops, the algorithm never selects the same node twice. Therefore, if a candidate is already an intermediate node on the path, the algorithm selects the next eligible candidate. When the number of CAN nodes grows, zones become rectangular areas of different sizes. In this scenario, there is an increasing probability that the algorithm will become trapped in a zone where all neighbor nodes are already intermediate nodes on the path. In such a case the routing algorithm enters a recovery mode that forces the JOIN message to backtrack its steps one at a time. For each step back all the neighbors of the node at that particular step are checked for an alternative path towards the destination. The recovery mode continues until a valid path is found.

Efficient routing is a critical aspect of every CAN implementation. Today, a number of CAN routing algorithms are available. The goal of our research is to implement and evaluate the performance of several important algorithms. To the best of our knowledge, the routing algorithms can be partitioned as follows [4], [14]–[16]:

- Pythagorean based algorithm
- Greedy forwarding along the x- and y-axes
- Greedy forwarding with shortcut nodes
- Inclination angle based algorithms
- Binary based routing

Out of these, we have implemented and evaluated the performance of the first four algorithms, which are reported in our paper.

A. CAN routing algorithm 1: Pythagorean based algorithm

This algorithm utilizes the Pythagorean theorem, calculating the shortest distance (hypotenuse) to the destination, as shown in Algorithm 2. This is done for each and every node along the path, only selecting the nodes with the shortest distance towards the destination as next hop. Note, however, that the Pythagorean based straightest path is not necessarily the shortest path.

Algorithm 2 CAN routing algorithm 1

1: **procedure** ROUTE(c, P)
Route JOIN message through CAN towards point P via node c, return owner p of point P
2: **if** $P \in$ c **then** ▷ Is P in c's neighbors n
3: $p \leftarrow n$ ▷ Set n as current node p
4: **else** ▷ P is not in origo node c's zone
Owner of zone where point P lie needs to be found
5: $p \leftarrow c$ ▷ Current node is set to p
6: **while** $P \neq$ p **do** ▷ Until owner to P is found
Check all neighbors n of current node p for shortest distance to point P
7: $d \leftarrow sqrt((Px2 - nx2) + (Py2 - ny2))$
Px,Py is x,y for point P, nx,ny is x,y for neighbor n
Neighbor n with shortest distance d is next hop on path to destination
8: $p \leftarrow n$ ▷ New current node p
9: **end while**
10: **end if**
11: Point P is in current node p's zone, return p
12: **return** p
13: **end procedure**

B. CAN routing algorithm 2: Greedy forwarding along the x- and y-axes

This algorithm routes along the x and y dimensions towards the destination, using the identifiers (x,y coordinates) of the bootstrap node and the joining node respectively, as shown in Algorithm 3. Reaching the destination becomes a matter of continuously evaluating the delta value, d, while progressing first along the x and then y dimensions until the destination identifier is reached.

C. CAN routing algorithm 3: Greedy forwarding with shortcut nodes

This algorithm performs the routing operation in much the same way as the previous algorithm, however with an important difference. Here the notion of shortcut zones is introduced, significantly decreasing the path length by minimizing the routing distances, as shown in Algorithm 4. The CAN Cartesian space is equally shared among four nodes in every corner of the grid. Each node is responsible for a quarter of the Cartesian space. Depending on the identifier of the new node joining the CAN overlay, the bootstrap node (keeping a list of all the current shortcut nodes) sends the new joining node to the shortcut node with the coordinates closest to the joining node identifier.

Algorithm 3 CAN routing algorithm 2

procedure ROUTE(c, P)
Route JOIN message through CAN towards point P via node c, return owner p of point P
  if $P \in$ c **then** ▷ Is P in c's neighbors n
    $p \leftarrow n$ ▷ Set n as current node p
  else ▷ P is not in origo node c's zone
Owner of zone where point P lie needs to be found
    $p \leftarrow c$ ▷ Current node is set to p
    while $P \neq$ p **do** ▷ Until owner to P is found
      while $Px \neq$ nx **do** ▷ Find P's x value
Check all neighbors n of current node p for shortest distance to point P's, x coordinate
        $d \leftarrow (Px - nx)$ ▷ Px, is x for point P
Neighbor n with shortest distance d is next hop on path to destination
        $p \leftarrow n$ ▷ New current node p
      end while
      while $Py \neq$ ny **do** ▷ Find P's y value
Check all neighbors n of current node p for shortest distance to point P's, y coordinate
        $d \leftarrow (Py - ny)$ ▷ Py, is y for point P
Neighbor n with shortest distance d is next hop on path to destination
        $p \leftarrow n$ ▷ New current node p
      end while
    end while
  end if
  Point P is in current node p's zone, return p
  return p
end procedure

Algorithm 4 CAN routing algorithm 3

procedure ROUTE(c, P)
Route JOIN message through CAN towards point P via node c, return owner p of point P
  if $P \in$ c **then** ▷ Is P in c's neighbors n
    $p \leftarrow n$ ▷ Set n as current node p
  else ▷ P is not in origo node c's zone
Check shortcut nodes s of origo node c to find the one closes to point P, set closest as new current node
    $p \leftarrow s$ ▷ Current node is set to p
    while $P \neq$ p **do** ▷ Until owner to P is found
      while $Px \neq$ nx **do** ▷ Find P's x value
Check all neighbors n of current node p for shortest distance to point P's, x coordinate
        $d \leftarrow (Px - nx)$ ▷ Px, is x for point P
Neighbor n with shortest distance d is next hop on path to destination
        $p \leftarrow n$ ▷ New current node p
      end while
      while $Py \neq$ ny **do** ▷ Find P's y value
Check all neighbors n of current node p for shortest distance to point P's, y coordinate
        $d \leftarrow (Py - ny)$ ▷ Py, is y for point P
Neighbor n with shortest distance d is next hop on path to destination
        $p \leftarrow n$ ▷ New current node p
      end while
    end while
  end if
  Point P is in current node p's zone, return p
  return p
end procedure

D. CAN routing algorithm 4: Inclination angle based algorithms

This algorithm routes according to a specific concept based on the inclination (angle), as shown in Algorithm 5. The angle between the bootstrap node from where the routing starts and the destination node is first calculated. Then, the angle to the destination of every node along the path is computed and compared with the original start to destination angle. The node with the destination angle closest to the starting angle becomes the next hop on the path towards the destination. By selecting the next hop with the smallest difference in angle compared to the original starting angle, the straightest path to the destination should be achieved. Observe however that the straightest path does not necessarily mean the shortest path.

V. ROUTING PERFORMANCE

One of the most fundamental parameters deciding the CAN routing performance is how often the routing algorithm enters the recovery mode. This in turn, influences several important parameters like, e.g., routing pathlength, routing time, and scalability performance. In order to evaluate this, we have done a number of experiments and evaluated the mean number of routing failures per node. Accordingly, two experiments have been done for each of the four implemented routing algorithms. In both experiments we start with an empty CAN. Then we add up to 100 nodes to the CAN in the first experiment, and up to 1000 nodes to the CAN in the second experiment. For each added node n, we keep a variable a_n that counts the number of times the routing algorithm enters the recovery mode for that particular node. We run each experiment 100 times and compute the average number of times each node n enters the recovery mode. We call this statistic *mean number of routing failures per node* n and we denote it by A_n. More formally:

$$A_n = \frac{1}{100} \sum_{r=1}^{100} a_{n,r} \quad , \tag{1}$$

where $a_{n,r}$ indicates the number of times the node n enters recovery mode during simulation run r.

The figures 1 and 2 show the mean number of routing failures per node for 100 nodes and 1000 nodes, respectively. It is observed that the variation of the plots is highly fluctuative.

Algorithm 5 CAN routing algorithm 4

procedure ROUTE(c, P)
Route JOIN message through CAN towards point P via node c, return owner p of point P
  if $P \in$ c **then** ▷ Is P in c's neighbors n
    $p \leftarrow n$ ▷ Set n as current node p
  else ▷ P is not in origo node c's zone
Owner of zone where point P lie needs to be found
    $dx \leftarrow (Px - cx)$ ▷ x for c and P
    $dy \leftarrow (Py - cy)$ ▷ y for c and P
    $d \leftarrow (dy/dx)$ ▷ Angle for c and P
    $p \leftarrow c$ ▷ Current node is set to p
    while $P \neq$ p **do** ▷ Until owner to P is found
Check all neighbors n of current node p for closest angle to destination angle d
      $ax \leftarrow (Px - nx)$ ▷ x for n and P
      $ay \leftarrow (Py - ny)$ ▷ y for n and P
      $a \leftarrow (ay/ax)$ ▷ Angle for n and P
Neighbor with angle a closest to angle d is next hop on path to destination
      $p \leftarrow n$ ▷ New current node p
    end while
  end if
  Point P is in current node p's zone, return p
  return p
end procedure

This is an indication of how well the algorithms are performing. Low values indicate that fewer errors are encountered while routing through CAN. If the sought node is found without failure, then this makes the curve plummet towards zero, thus explaining the appearance of dips in the graphs. This means that low values are indicative of a well performing algorithm, where many nodes encounter few errors. On the other hand, a plot with high values indicates a large amount of nodes encountering errors, indicating bad routing performance.

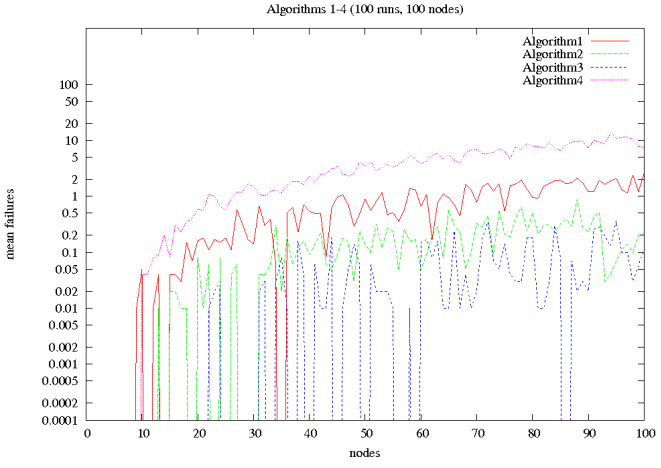

Fig. 1. Mean number of routing failures per node for 100 nodes

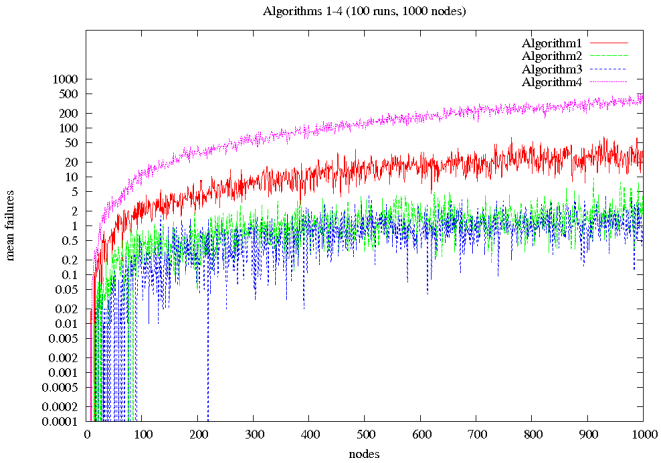

Fig. 2. Mean number of routing failures per node for 1000 nodes

We note that the performance of algorithms 1 (Pythagorean based algorithm) and 4 (Inclination angle based algorithms) are highly dependent on the number of nodes. This is an indication of scalability problems. On the other hand, the algorithms 2 (Greedy forwarding along the x- and y-axes) and 3 (Greedy forwarding with shortcut nodes) show good routing performance and, implicitly, good scalability.

TABLE I

Algorithm	**1**	**2**	**3**	**4**
$A_{\max}(100)$	2.6	0.9	0.4	13.6
$A_{\max}(1000)$	66.2	9.3	4.3	520
Increase factor	25.6	10.6	11.9	38.3

Table I shows the maximum of the mean failures encountered by any node

$$A_{\max}(N) = \max\{A_1, \ldots, A_N\} \quad (2)$$

where N denotes the maximum number of nodes in the corresponding experiment. By comparing $A_{\max}(N)$ for the different routing algorithms, we immediately recognize that algorithms 2 and 3 display much smaller failures than algorithms 1 and 4. A badly performing routing algorithm leads to a large increase in the number of error messages sent back and forth. In other words a frequently entered recovery mode entails a high degree of mean failures per node.

By having a closer look at the scaling behaviour of the algorithms observed in Table I, we can conclude as follows. In the case of a tenfold increase in the number of nodes (i.e., from $N = 100$ to $N = 1000$), the scalability performance is different depending upon the particular algorithm. For algorithm 1 and 4 a tenfold increase in the number of nodes, entails a respective 26 and 38-fold increase in the maximum mean failures, indicating clear scalability problems. Thus leading to the conclusion, that very few nodes can be accommodated by a CAN employing such a routing algorithm, before a total failure of the network is encountered. On the other hand, for the same increase in node number, algorithms 2 and 3 show

good scalability opportunities with a roughly 10-fold increase in the maximum of mean failures.

Furthermore, the constatation that algorithms 2 and 3, outperform algorithm 1 and in particular algorithm 4 is also confirmed by the observed simulation run times. When $N = 1000$ for algorithms 2 and 3, the run times are in the order of minutes, though for algorithms 1 and 4, being in the order of hours and days, respectively.

VI. Conclusions

Content Addressable Networks are an important part of the overlay approach to network virtualization. In this paper, we have reported on the implementation and performance evaluation of several CAN routing algorithms for CAN construction in a virtual space. These are the Pythagorean based algorithm, greedy forwarding along the x- and y-axes, greedy forwarding with shortcut nodes and inclination angle based algorithms. Their performance has been evaluated in terms of mean failure rate per node and scalability index. These parameters are highly decisive in the implementation of CAN networks.

Our results show that greedy based algorithms perform much better in a CAN network than the Pythagorean based algorithm and inclination angle based algorithm, although the last two algorithms follow the straightest path in a mathematical sense. These results are also confirmed by the simulations run times which are correlated with the performance of the specific routing algorithm, e.g., the more routing errors encountered the longer simulation times for the same number of simulated node joins. Furthermore, as a next step in our future work we plan on performing several more simulations with an increased number of added nodes. This will allow us to better estimate the scalability for each specific algorithm, leading to a more realistic approximation. In the future we plan to move the CAN implementation into a simulator such as MyNS [11] or OmNeT++ [12]. Through this we will gain access to the facilities offered by a complete simulation environment making it possible to study other relevant parameters, e.g., the CAN's performance in the presence of large amounts of churn, scalability, memory requirements. We are also planning to implement the remaining binary routing algorithm for CAN routing and compare the CAN performance to the performance of other structured-based overlays such as Pastry and Tapestry. Present with the opportunity to investigate the different pros and cons of CAN we can compared to other structured overlays and routing algorithms. Another important future work is to implement the straightest path not only in the virtual space but also in a real geographical space, taking into account several important demands such as minimum end-to-end routing delay.

References

[1] Balakrishnan H., Kaashoek F. M., Karger D., Morris R. and Stoica I., *Looking up data in P2P systems*, Communications of the ACM, Vol. 46, No. 2, pp. 43-48, February 2003.

[2] Dabek F., Zhao B., Druschel P., Kubiatowicz J. and Stoica I., *Towards a Common API for Structured Peer-to-Peer Overlays*, 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, CA, USA, February 2003.

[3] Dragovic B., Fraser K., Hand S., Harris T., and A Ho, *Xen and the art of virtualization*, SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003, Bolton Landing, NY, USA

[4] Eberspaecher J., Schollmeier R., Zöls S. and Kunzmann G., *Structured P2P Networks in Mobile and Fixed Environments*, International Workshop on Heterogeneous Networks HET-NETs 2004, UK

[5] Francis P., *YOID: Extending the Internet Multicast Architecture*, unpublished paper, http://www.isi.edu/div7/yoid/docs/yoidArch.ps.gz, April 2000.

[6] Gummadi K. P., Gummadi R., Gribble S. D., Ratnasamy S., Shenker S. and Stoica I., *The Impact of DHT Routing Geometry on Resilience and Proximity*, ACM SIGCOMM, Karlsruhe, Germany, August 2003.

[7] Ilie D. and Popescu Adrian, *A Framework for Overlay QoS Routing*, 4th Euro-FGI Workshop, Ghent, Belgium, May 2007.

[8] Kossmann D., *The state of the art in distributed query processing*, ACM Computing Surveys, Vol. 32, No. 4, pp. 422-469, December 2000.

[9] Loguinov D., Casas J. and Wang X., *Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience*, IEEE/ACM Transactions on Networking, Vol. 13, No. 5, pp. 1107-1120, October 2005.

[10] Lua E. K., Crowcroft J., and Pias M., Sharma R. and Lim S., *A Survey and Comparison of Peer-to-Peer Overlay Networks Schemes*, IEEE Communications Surveys and Tutorials, Vol. 7, No. 2, pp. 72-93, 2nd Quarter 2005.

[11] *Myns simulator*, http://www.cs.umd.edu/users/suman/research/myns/index.html

[12] *OmNeT++ simulator*, http://www.omnetpp.org

[13] Plaxton C. G., Rajaraman R. and Richa A. W., *Accessing Nearby Copies of Replicated Objects in a Distributed Environment*, ACM SPAA, June 1997.

[14] Popescu Alex, Ilie D. and Kouvatsos D., *On the Implementation of a Content-Addressable Network*, 5th International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs), Karlskrona, Sweden, February 2008.

[15] Ratnasamy S., Francis P., Handley M., Karp R. and Shenker S., *A Scalable Content-Addressable Network*, ACM SIGCOMM, San Diego, CA, USA, August 2001.

[16] Ratnasamy S., Shenker S. and Stoica I., *Routing Algorithms for DHTs: Some Open Questions*, 1st International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, USA, 2002.

[17] Risson J. and Moore T., *Survey of Research towards Robust Peer-to-Peer Networks: Search Methods*, RFC 4981, http://www.ietf.org/rfc/rfc4981.txt, September 2007.

[18] Yang B. and Garcia-Molina H., *Efficient Search in Peer-to-Peer Networks*, ICDCS, Vienna, Austria, July 2002.