A Scalable Content-Addressable Network [1]
S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker

Proseminar “Algorithmen für Rechnernetze”
Lehrstuhl für Rechnernetze und Telematik,
Prof. Dr. Christian Schindelhauer

Lukas Veneziano
Motivation

Regular client-server structures

**Pros**
- Easy to use for the user
- Easy to maintain

**Cons**
- High traffic
- Low fault tolerance
- Single point of failure
- Scalability
- High costs

Client-server communication [Graphic 1]
Motivation

- Regular peer-to-peer communication

**Pros**
- Low traffic for server
- High speed for peer
- Better fault tolerance

**Cons**
- Find other peers
- Server needed
Motivation

Distributed hash table network (DHT)

- **Pros**
  - Low traffic for server
  - High speed for peer
  - High fault tolerance
  - Self-organizing

- **Cons**
  - Mostly Usersoftware

Peer-to-peer with DHT communication [Graphic 1]
Motivation

- Distributed hash table network (DHT)
  - Four DHT proposals (CAN, Chord, Pastry, Tapestry)
  - Unlike Napster etc.
    - Usable as back-end
    - Search for Data in the network
  - No server needed
Motivation

- Content-Addressable Network (CAN)
  - Distributed, decentralized P2P infrastructure
  - High scalability
  - High fault tolerance
  - Easy to search (hash-table)
Overview

1. Design
   - Routing
   - Construction
   - Maintenance

2. Improvements

3. Measurement data
Design

- Content Addressable Network
  - Mapped to area of $[0,1]^n$
  - Organized with hash-table

Concept of a hash-algorithm
Design

- Content Addressable Network
  - Data is stored in zones
    - Each node represents one zone
  - N-dimensional torus
    (not visible in graphic)
  - Paths are short because of torus structure
Design

- Content Addressable Network
  - Expected number of hops for CAN in d-dimensions: \( O\left(\frac{d}{\sqrt{n}}\right) \)
  - Average degree of a node: \( O(d) \)
Design

- One node
  - Knows its zone
  - Knows its neighbours
  - Knows the specification of the specific CAN
  → Can act autonomic

Formula for hops and degree from [2]
Overview

1. Design
   - Routing
   - Construction
   - Maintenance

2. Improvements

3. Measurement data
Routing

- Simple routing along the axes
  - e.g.
    1. X-axis
    2. Y-axis
    3. Z-axis
Overview

1. Design
   - Routing
   - Construction
   - Maintenance

2. Improvements

3. Measurement data
Construction

- One node is a CAN
- Until other nodes are added

A CAN which consists of only one node
Add other nodes

1. Find a node in the CAN
Construction

- One node can be a CAN
- Add other nodes
  1. Find a node in the CAN
  2. Route to a random zone

A CAN of 8 nodes
Construction

- One node can be a CAN
- Add other nodes
  1. Find a node in the CAN
  2. Route to a random zone
     a. Split the random zone

![Diagram of a CAN with new node addition](image-url)
Construction

- One node can be a CAN
- Add other nodes
  1. Find a node in the CAN
  2. Route to a random zone
     a. Split the random zone
  3. Inform the neighbours

A CAN of 9 nodes
Construction

Store Data in a CAN

- Find a node
- Transfer data into key
- Route data by key to zone
Overview

1. Design
   - Routing
   - Construction
   - Maintenance

2. Improvements

3. Measurement data
Maintenance

- Each node sends alive messages to its neighbours
Maintenance

- Each node sends alive messages to its neighbours
- Takeover of dead nodes
  - Takeover message

A CAN of 8 nodes
Maintenance

- Each node sends alive messages to its neighbours
- Takeover of dead nodes
  - Takeover message
Overview

1. Design
   - Routing
   - Construction
   - Maintenance

2. Improvements

3. Measurement data
Improvements

- Many improvements for speed and fault tolerance for the CAN-concept
  - More dimensions
  - Redundant coordinate-spaces
  - Routing with IP-topology
  - Node-allocation with IP-topology
  - Node-allocation with information of scope
  - Zone is managed by multiple nodes
  - Multiple hash-algorithms
  - Caching
Overview

1. Design
   - Routing
   - Construction
   - Maintenance

2. Improvements

3. Measurement data
Measurement data

- Used metrics
  - Path length
  - Neighbour Count
  - Latency
  - Volume
  - Fault tolerance
  - Hash table availability
### Measurement data - parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>“bare bones” CAN</th>
<th>“knobs on full” CAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimensions</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>realities</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>peers per zone</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td># hash functions</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RTT weighted routing metric</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>uniform partitioning</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>landmark ordering</td>
<td>OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Data from [1]
### Measurement data - results

<table>
<thead>
<tr>
<th>Metric</th>
<th>“bare bones” CAN</th>
<th>“knobs on full” CAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path length</td>
<td>198.0</td>
<td>5.0</td>
</tr>
<tr>
<td># neighbours</td>
<td>4.57</td>
<td>27.1</td>
</tr>
<tr>
<td># peers</td>
<td>0</td>
<td>2.95</td>
</tr>
<tr>
<td>IP Latency</td>
<td>115.9ms</td>
<td>82.4ms</td>
</tr>
<tr>
<td>CAN Latency</td>
<td>23,008ms</td>
<td>135.29ms</td>
</tr>
</tbody>
</table>

data from [1]
Conclusion

- powerful technique to store data in an internet-like scale
- much more scalable, fault tolerant and faster than a regular server-structure
- with some improvements even more powerful
Thank you for your attention
References