Search in peer-to-peer systems

Leonardo Querzoni

5-6-2008
Peer-to-peer systems

- **What is a peer-to-peer system?**
  - A set of hosts (nodes) that communicate and exchange information to implement a service/application
  - Usually all the hosts have the same role in the system
  - Results stem from collaboration among nodes

- **Common characteristics:**
  - Large scale
    - number of nodes
    - global load
  - Nodes unreliability
    - Faults
    - Churn
    - Selfish/Byzantine behaviors
  - Lack of central administration => self-administration
Search in P2P systems

- **Problem**: the user must access some data stored somewhere in the system

- The search mechanism, given a query, returns data satisfying the query or identifiers of nodes where these data are stored.

- **Challenges**:
  - Nodes are unreliable: they can exit the system at any time, due to crash or leave
  - The scale of the system is large
    - searched data can be stored on one single node among millions
    - millions of users also mean a large number of concurrent searches
Functional requirements

■ Expressiveness
  ■ The way the system let user specify queries.

■ Examples:
  ■ **Key lookup**: each resource is identified with a “key” (identifier). A search specifies a key. The mechanism returns data identified by the searched key.
  
  ■ **Keywords query**: a query contains a set of searched keywords (e.g. Q="low-cost flight Paris"). The mechanism returns data containing the searched keywords. Results of keyword searches can be sometimes ranked and filtered by relevance.
  
  ■ **Aggregate**: aggregate functions (Sum, Count, Max, etc.) can be used to obtain properties extracted from the whole data collection (e.g. How many documents contains the world “p2p”?)
  
  ■ **SQL**: Current research on supporting SQL in P2P systems is preliminary.
Functional requirements

- **Stop condition**
  - Defines the “completeness” of the answer expected from the system
    - every result
    - the first result found
    - a subset of the results with a certain size
Non-functional requirements

■ **Quality of Service**
  
  User-perceived qualities.

  ■ **Accuracy** - the result set does not contain data that do not match the query

  ■ **Result set size**
    
    ■ **Completeness** - the result set contains all data that satisfies the search criterion
    
    ■ **Satisfaction** - the first $n$ results are provided to the user
    
    ■ A single result is sufficient

  ■ **Responsiveness**
    
    ■ Time to obtain the first result
    
    ■ Time to obtain the entire set of results

  More specific QoS metrics can depend on applications.
Non-functional requirements

- **Efficiency**
  - Reduced resource usage to answer a search
    - Bandwidth
    - Memory
    - CPU
    - Battery
    - ...

- **Robustness**
  - Failures and dynamics should not impact service efficiency and QoS
Design choices

- **System topology**
  - Tree, ring, torus, mesh, butterflies, random graphs, k-regular graphs, etc.

- **Data positioning**
  - **Source**
    - Data is left on the node that inserted it in the system
  - **Structure**
    - Data (or a pointer) is moved to a specific node
  - **Replicated (with structure)**
  - **Replicated (random)**
    - Replication increases both robustness and responsiveness
Design choices

- **Routing**
  - How queries are routed inside the system to reach those nodes where they are positively evaluated

- **Blind search approach**
  - No hint about the position of searched data

- **Flooding-based techniques**
  - simple
  - robust
  - expensive

- **Random walks**
Routing
- How queries are routed inside the system to reach those nodes where they are positively evaluated

Informed search approach
- Information about data positioning is first distributed in the system and then exploited to improve search mechanism performance

Exact search
- Like in DHTs

Hint-based
- Approximate information about data location
A small survey: blind search

- **Breadth First Search (BFS) / Flooding**
  - No specific data positioning strategy
  - The query is routed to all the nodes in the network
  - No limits on expressiveness
  - Huge overhead => low efficiency
  - Robust
  - Returns complete results
A small survey: blind search

- **Limited horizon**
  - No specific data positioning strategy
  - The query is flooded in the network
  - The search scope is limited through Time-To-Live.
  - No limits on expressiveness
  - Robust
  - Completeness of the result set cannot be guaranteed
  - Goal: reduce query forwarding
  - [Gnutella, www.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf]
A small survey: blind search

- **Iterative deepening**
  - Similar to limited horizon
  - Various runs with increasing TTL values until results are found
  - Suited to applications where a subset of results is needed
  - Goal: reduce query forwarding in most cases while still providing result completeness
  - [Yang, Garcia-Molina - "Improving Search in Peer-to-Peer Networks", ICDCS ‘02]
**A small survey: blind search**

- **Directed BFS**
  - Each node forward a query to a subset of neighbors
  - The query is sent to those neighbors through which nodes with many quality results may be reached, thereby maintaining quality of results
    - Neighbors that returned an high number of results for previous queries
    - Neighbors that returned response messages that have taken the lowest average number of hops (searched data is in its proximity)
    - Neighbors that has forwarded large number of messages (stable node with available resources)
    - Neighbors with short message queues (avoid congestion)
  - Queries are routed from the “edges” of the system toward more “populated” zones
  - Goal: reduce query forwarding
  - [Yang, Garcia-Molina - "Improving Search in Peer-to-Peer Networks", ICDCS ‘02]
A small survey: blind search

- **Local indices**
  - Routing is done like in BFS
  - Each node maintains information about data stored on nodes that are \( r \)-hops away from it
  - Each query is evaluated only at specific distances from the source
  - Nodes that evaluate the query answer on behalf of those nodes whose data they index
  - Goal: reduce global processing time
  - [Yang, Garcia-Molina - "Improving Search in Peer-to-Peer Networks", ICDCS '02]
A small survey: blind search - cache based

- **Uniform index caching (UIC)**
  - Equivalent to limited horizon but...
  - Each node maintains in a cache answer to the last $n$ query answers that it forwarded
  - Queries contained in the cache are answered quickly, without the need of forwarding them.
  - Problems related to cache management
  - Sensible to topology changes
  - Goal: improve responsiveness and reduce query forwarding
A small survey: blind search - cache based

**Distributed caching and adaptive search (DiCAS)**

- UIC causes a lot of duplicate cache entries in the whole system
- In DiCAS each node caches only specific queries
  - each node choose a random group identifier \([1,\ldots,x]\) when it joins the system
  - each query is mapped to a group identifier through an hash function
  - a node caches a query result only if the query hash matches its group identifier
- Queries are mainly routed to nodes that can have an answer in their caches
- Goal: like UIC, but also reduces memory overhead due to caching
- [Wang, Xiao, Liu, Zheng - "Distributed Caching and Adaptive Search in Multilayer P2P Networks", ICDCS ‘04]
Distributed caching and adaptive search (DiCAS)

For a passing query response, each peer overhearing will cache this response, as illustrated in Figure 5. Only the peers in group 1 will receive the response independently performs a computation on the file name's hash value equals 1. Only the peers in group 1 will perform the response and search the desired file name in the query. For example, when a query is forwarded to matched neighbors in most cases. However, it is still possible that query forwarding can be blocked if none of a peer's neighbors have a group ID that matches the hash value of a query. The peer will select a neighbor with the highest connectivity degree to forward the query to in this case. Based on the group ID's uniform distribution, a peer will only be forwarded to neighbors with group ID 1. We claim the group ID is uniformly distributed in the modulus operation, the whole network is logically divided into M groups. A uniform hash algorithm is employed to translate the queried file name string to a hash value. We define that a query matches a peer if the equation below is satisfied.

\[ \text{Peer Group ID} = \text{hash(query)} \mod M \]

Benefiting from the group ID's uniform distribution, a distributed caching and adaptive search (DiCAS) system ensures that duplicated query responses among neighboring peers can be minimized. In DiCAS, when a peer joins the P2P system, it will randomly take an initial value in a certain range of 1. We claim the group ID is uniformly distributed in the modulus operation result of a new peer. Suppose the modulus operation result of a new peer is 1. Only the peers in group 1 will forward the query to peers with the matched group ID. Those peers form a virtual layer which has much smaller connectivity and traffic volume compared to the original P2P network due to its value being randomly chosen.

5.1. Considerations of P2P Simulation

In this section, we present our simulation methodology to investigate searching and index caching on all peers of DiCAS for the purpose of evaluating the performance of peers in the Gnutella network configured with the support of the DiCAS mechanism. We choose to simulate each peer's message-level behaviors as an effort to understand the behavior of a large-scale cache-aware P2P network. We choose to simulate the environments of real-world P2P networks to ensure that simulation results and observations are reasonable. For such a scenario, one should carefully choose related parameters and distributions such as network and topology parameters, workload parameters, and initial configuration. Moreover, the performance of the underlying approach. Thus, the problem of choosing a distribution for the request size distribution is one of the most important issues to consider. The parameters that determine the simulation scenario are user's query length distribution, the request distribution, request frequency distribution, and the network topology. The request distribution does not follow the strict Zipf distribution but with a heavy tail. The observed value of the exponent varies from trace to trace. The request distribution does not follow the Zipf distribution but with a heavy tail.

The length of time that nodes remain available follows a power law. The request length distribution does not follow a Zipf distribution. User's query length distribution follows a heavy-tailed distribution. Query word frequency distributions over the network. Content popularity at a publisher follows Zipf-like distribution (aka Power Law) where the relative probability of a page is proportional to its popularity. Query word frequency over the network. Content popularity at a publisher follows Zipf-like distribution (aka Power Law). Query word frequency distributions follow a heavy-tailed distribution. Query word frequency distributions over the network. Content popularity at a publisher follows Zipf-like distribution (aka Power Law). Query word frequency distributions follow a heavy-tailed distribution. Query word frequency distributions over the network. Content popularity at a publisher follows Zipf-like distribution (aka Power Law). Query word frequency distributions follow a heavy-tailed distribution.

The observed value of the exponent varies from trace to trace. The request distribution does not follow the Zipf distribution but with a heavy tail. The observed value of the exponent varies from trace to trace. The request distribution does not follow the Zipf distribution but with a heavy tail. The observed value of the exponent varies from trace to trace. The request distribution does not follow the Zipf distribution but with a heavy tail.
A small survey: blind search

- **Random walks**
  - Simple: each node forwards the query to one of its neighbors
  - Stop condition: depends on the number of desired results
  - Needs STRONG data replication or caching to deliver acceptable robustness and responsiveness levels.
  - Goal: reduce query forwarding while quickly adapting to topology changes
A small survey: Informed search

**DHTs**

- Nodes constituting the system are organized in order to realize a distributed implementation of an hash table
  - each node is responsible for maintaining a specific subset of data
  - an intelligent query routing mechanism route the query to the destination node in a logarithmic number of steps
  - the mapping between data and nodes is realized through a globally known consistent hash function

- Elegant and efficient solution for exact-match queries => limited expressiveness

- Some solutions are sensible to churn


- [A. Rowstron and P. Druschel, Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems, Proceedings of International Conference on Distributed Systems Platforms (Middleware), 2001](#)


A small survey: Informed search

- **Hybrid systems - Kelips**
  - DHT-like infrastructure with mixed topologies:
    - high level - structured
    - low level - random
  - Based on gossip algorithms (probabilistic approach)
  - Self-stabilizing properties
  - More resistant to churn/failures
  - Constant cost for searches
  - [K. Birman et al., 2007]
A small survey: Informed search

**Routing Indices**

- Each index contains information about the direction each query should take for the next hop
- Information is based on data stored on nodes, and is aggregated to reduce memory footprint
- Information aggregation makes RIs “coarser” than local indices => routing is not “exact”
- Requires data stored on nodes to be classified (e.g. with topics)
- Resemble event routing infrastructures for pub/sub
- [Crespo, Garcia-Molina - "Routing Indices for Peer-to-peer Systems", ICDCS '02]
A small survey: Informed search

**Probabilistic routing tables**

- Routing tables contain approximated information about data positioning.
- Information is spread from the data source and recorded in routing tables using exponentially decaying bloom filters.
- The farther a node is from the data source, the more approximated the information stored in its routing table is.

[Kumar, Xu, Zegura - "Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks", INFOCOM '05]
Performance evaluation

- Test application: P2P telephone service
- Each user stores its profile in the system
- The profile can be replicated
- Each profile is identified by a unique userID
- A user can lookup a profile on the basis of a userID
- Users can enter or leave the system at any time depending on their characterizing class
## Performance evaluation

### User classes:

<table>
<thead>
<tr>
<th>Profile</th>
<th>ADSL</th>
<th>Wi-Fi</th>
<th>UMTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Desktop</td>
<td>Long session</td>
<td>Short session</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Session Time</th>
<th>Min(h)</th>
<th>Avg(h)</th>
<th>Profile</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min(h)</td>
<td>∞</td>
<td>0.5</td>
<td></td>
<td>∞</td>
<td>0.083</td>
<td>0.5</td>
<td>0.083</td>
<td></td>
</tr>
<tr>
<td>Avg(h)</td>
<td>6</td>
<td>0.83</td>
<td></td>
<td>1</td>
<td>1.09</td>
<td>1</td>
<td>1.09</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>&lt;1</td>
<td>1.09</td>
<td></td>
<td>1</td>
<td>1.09</td>
<td>1</td>
<td>1.09</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Max Down/Up</th>
<th>Profile</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20M / 384kbps</td>
<td></td>
<td>11M</td>
<td>11M</td>
<td>384kbps</td>
<td>384kbps</td>
<td></td>
</tr>
</tbody>
</table>
Performance evaluation

SCENARIO 1

<table>
<thead>
<tr>
<th>Profile</th>
<th>Session Time (avg h)</th>
<th>Inter-arrival Time (sec)</th>
<th>% population</th>
<th>Nodes online (avg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>0</td>
<td>20%</td>
<td>20.000</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>2.18</td>
<td>5%</td>
<td>5.000</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0.18</td>
<td>5%</td>
<td>5.000</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>0.73</td>
<td>45%</td>
<td>45.000</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>0.18</td>
<td>25%</td>
<td>25.000</td>
</tr>
</tbody>
</table>
### Performance evaluation

#### SCENARIO 2

<table>
<thead>
<tr>
<th>Profile</th>
<th>Session Time (avg h)</th>
<th>Inter-arrival Time (sec)</th>
<th>% population</th>
<th>Nodes online (avg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>0</td>
<td>20%</td>
<td>20.000</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>2,18</td>
<td>10%</td>
<td>10.000</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0,18</td>
<td>20%</td>
<td>20.000</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>0,73</td>
<td>30%</td>
<td>30.000</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>0,18</td>
<td>20%</td>
<td>20.000</td>
</tr>
</tbody>
</table>
Performance evaluation

We tested three different systems:

- Cyclon (unstructured)
- Kademlia (structured)
- Kelips (hybrid)
- nCyclon
Cyclon is a distributed overlay management protocol:

- Builds and maintains a logical network connecting a set of nodes
- This network resembles a random graph
  - Strong connectivity
  - Small diameter

It is based on the "view exchange" mechanism:

- Each node maintains a small view (set of neighbors) that is a subset of the entire system population.
- Periodically a node exchange a part of its view with one of its neighbors

At runtime each local view can be considered as a uniform random sample of the entire system population.
Cyclon

An example:
An example:
Cyclon

An example:
An example:
Cyclon

An example:
Cyclon does not provide any primitives for data storage/retrieval.

Profile storage is done by the same profile owner (no replication)

Search is implemented with a simple flooding-based broadcast

- The searching node sends a lookup message to all its neighbors
- Each node that receives a lookup message:
  - discards the message if it previously received another message for the same lookup
  - reply back if it stores the searched profile
  - Forwards the message to all its neighbors
Kademlia is an algorithm implementing a distributed hash table.

- Every node is identified by a 160 bit unique NodeID.
- Every object, identified by a 160 ObjectID, is managed by the K nodes whose NodeID is closer to ObjectID.
- K is a system-wide parameter that defines the level of replication for each object.
- Distance between IDs is obtained applying the XOR operator.
- Object search is realized through either a recursive or iterative distributed algorithm. The algorithm at each step get closer to the node holding the searched object with a logarithmic evolution.
- Object search is realized exploiting local data structures (Buckets) that store references to other nodes in the system.
Kademlia

Search is executed through successive approximations of the target \textit{ObjectID}
Service implementation on top of Kademlia:

- Each NodeID applying an hash function to the profile's friendly name of the corresponding user.

- K copies of the profile are stored on the K-1 nodes whose NodeIDs are closer to the user’s NodeID.

- To search for a profile a node just needs to know the friendly name associated to the profile.
Kelips is a distributed algorithm implementing a hybrid structured/unstructured scheme with store/lookup capabilities:

- Every node is mapped through a hash function to one of several groups.
- Every object, mapped to one of the groups, is managed by one of the nodes belonging to that group.
- Every node manages three data structures:
  - Local view: the set of nodes belonging to its group.
  - Contacts: a subset of nodes belonging to other groups.
  - Filetuples: an index of objects managed by nodes in its group.
- Data structure content is maintained up-to-date through a continuous gossip stream of information among nodes.
An example:

Group 1

Group 2

Group k
Object search:

- If the node group and searched object group match, the node just looks into filetuples.
- Otherwise it looks into contacts for a node belonging to the same group of the searched object and forward it the request.
- To improve robustness versus stale information in data structures, the node can forward randomly the request through a (short) random walk.
Service implementation on top of Kelips:

- The group a node pertains to is calculated applying a hash function to the user’s friendly name.
- The profile is maintained on the same user’s node.
- To search for a profile a node just needs to know the friendly name associated to the profile.
nCyclon

Idea:
- provide information on the location of data
- information is maintained through randomized algorithms
- search delivers probabilistically correct results

The system is built on top of:
- a Peer Sampling Service delivering uniform random samples of the system population (implemented with Cyclon)
Each node locally manages an Access Point Table (APT) to store pointers to some nodes which maintain replicas of each data object.

Each entry of the APT table is a tuple \((i, IP)\) where:

- \(i\) is the identifier of the data object
- \(IP\) is the IP address of a node which store a replica of object \(i\)

<table>
<thead>
<tr>
<th>Name</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>IP-B₈</td>
</tr>
<tr>
<td>Bob</td>
<td>IP-B₁₂</td>
</tr>
<tr>
<td>Carla</td>
<td>IP-B₁</td>
</tr>
</tbody>
</table>
Data object search:

- the user passes the object identifier $i$ as input to the search procedure
- the search is realized as a random walk in the system
- each node receiving the search message checks if its APT contains an entry $(i, IP)$.
  - true => forwards the search message to $IP$
  - false => forward the message to a random node

The random walk stops either if it reaches a node which stores the searched data object, or after a given number of steps (TTL)
Data object search:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bready</td>
<td>IP-B11</td>
</tr>
<tr>
<td>John</td>
<td>IP-B1</td>
</tr>
<tr>
<td>Dan</td>
<td>IP-B13</td>
</tr>
</tbody>
</table>

Alice | IP-B24  |
Dan   | IP-B13  |
Jack  | IP-B5   |

Jess  | IP-B4   |
Mark  | IP-B5   |
Lou   | IP-B8   |

Alice | IP-B8   |
Bob   | IP-B12  |
Carla | IP-B1   |

search(john)
The probability to retrieve a data object is constant and depends on:
- the length of the random walk
- the APT size
- the amount of stored objects

The probability is independent from the system size and the replication degree.

\[
Pr(\text{success}) = 1 - \left(1 - \frac{|APT|}{|data|}\right)^{TTL}
\]
Simulation environment

Event-driven simulation environment: J-Sim (http://www.j-sim.org)

- Real algorithm implementation
  - Kademlia, Cyclon, Kelips
  - Partial implementation of our solution (NCyclon):
    - Simplified storage
    - No adaptive replication
    - Robust retrieval
- Simulated communication substrate
  - direct point-to-point perfect links
  - adds latency (gaussian) to each message
- Simulated timeline
Simulation environment

Warmup + Measures

Population evol.

Number of nodes

Time (sec.)

Warmup

Measures
**Test setup**: 10,000 profiles, 1,000 nodes on-line, single class with simplified churn model.

![Cyclon - Node stress distribution chart]
Node stress distribution

**Test setup:** 10,000 profiles, 1,000 nodes on-line, single class with simplified churn model.
**Test setup:** 10,000 profiles, 1,000 nodes on-line, single class with simplified churn model.
Distribution of users on different classes:

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>20%</td>
</tr>
<tr>
<td>Class B</td>
<td>5%</td>
</tr>
<tr>
<td>Class C</td>
<td>5%</td>
</tr>
<tr>
<td>Class D</td>
<td>45%</td>
</tr>
<tr>
<td>Class E</td>
<td>25%</td>
</tr>
<tr>
<td>Class A</td>
<td>20%</td>
</tr>
<tr>
<td>Class B</td>
<td>10%</td>
</tr>
<tr>
<td>Class C</td>
<td>20%</td>
</tr>
<tr>
<td>Class D</td>
<td>30%</td>
</tr>
<tr>
<td>Class E</td>
<td>20%</td>
</tr>
</tbody>
</table>
Successful lookups (%)

**Test setup:** Scenario 1 and 2, 100,000 profiles, 10,000 nodes on-line

**Metric:** % successful lookups = \( \frac{\text{Num. profiles found}}{\text{Num. lookup operations on online profiles}} \)

### S1 - Successful lookups

<table>
<thead>
<tr>
<th></th>
<th>Cyclon</th>
<th>NCycloon</th>
<th>Kademlia</th>
<th>Kelips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>100.0%</td>
<td>96.0%</td>
<td>92.2%</td>
<td>82.2%</td>
</tr>
</tbody>
</table>

### S2 - Successful lookups

<table>
<thead>
<tr>
<th></th>
<th>Cyclon</th>
<th>NCycloon</th>
<th>Kademlia</th>
<th>Kelips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>99.7%</td>
<td>95.4%</td>
<td>91.7%</td>
<td>81.8%</td>
</tr>
</tbody>
</table>
**Test setup:** Scenario 1 and 2, 100,000 profiles, 10,000 nodes on-line

**Metric:** average number of messages per lookup

![Graph of Messages per lookup](image)

- **S1 - Messages per lookup:**
  - Cyclon: 195,692
  - NCyclon: 3.6
  - Kademlia: 39.6
  - Kelips: 3.0

- **S2 - Messages per lookup:**
  - Cyclon: 192,952
  - NCyclon: 3.8
  - Kademlia: 40.0
  - Kelips: 3.1
**Overhead for system management**

**Test setup:** Scenario 1 and 2, 100,000 profiles, 10,000 nodes on-line

**Metric:** number of messages needed for system management (excluding lookups)

---

### S1 - Overhead

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Millions of Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclon</td>
<td>17.97</td>
</tr>
<tr>
<td>NCyclon</td>
<td>27.79</td>
</tr>
<tr>
<td>Kademlia</td>
<td>11.20</td>
</tr>
<tr>
<td>Kelips</td>
<td>37.55</td>
</tr>
</tbody>
</table>

### S2 - Overhead

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Millions of Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclon</td>
<td>18.37</td>
</tr>
<tr>
<td>NCyclon</td>
<td>30.85</td>
</tr>
<tr>
<td>Kademlia</td>
<td>13.11</td>
</tr>
<tr>
<td>Kelips</td>
<td>40.56</td>
</tr>
</tbody>
</table>
**Test setup:** Scenario 1 and 2, 100,000 profiles, 10,000 nodes on-line

**Metric:** % of messages sent through stale links

### S1 - Churn sensibility

<table>
<thead>
<tr>
<th></th>
<th>Cyclon</th>
<th>NCyclon</th>
<th>Kademlia</th>
<th>Kelips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead links (%)</td>
<td>2.9%</td>
<td>4.2%</td>
<td>10.8%</td>
<td>17.2%</td>
</tr>
</tbody>
</table>

### S2 - Churn sensibility

<table>
<thead>
<tr>
<th></th>
<th>Cyclon</th>
<th>NCyclon</th>
<th>Kademlia</th>
<th>Kelips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead links (%)</td>
<td>3.4%</td>
<td>4.8%</td>
<td>11.2%</td>
<td>17.6%</td>
</tr>
</tbody>
</table>
How do these systems grow?

**Test setup:** 1k-10k-100k-1000k profiles, 10% nodes on-line, classes configured as in Scenario 1

**Metric:** % successful lookups

![Scalability Graph](image)
How do these systems grow?

Detail on Kademlia behaviour with 100k and 1M profiles (Scenario 1).

**S1 - Successful lookups**
- 100K: 92.17%
- 1M: 91.57%

**S1 - Messages per lookup**
- 100K: 39.6
- 1M: 62.8

**S1 - Overhead**
- 100K: 11
- 1M: 134

**S1 - Churn sensibility**
- 100K: 11%
- 1M: 5%
The presence of nodes that are always online impacts the behaviour of the system.

**Test setup:**

- 10,000 profiles
- 1,000 nodes online from class B
- Plus:
  - 100-1,000 nodes online from class B
  - 100-1,000 nodes online from class A
Influence of desktop nodes

**Metric:** % successful lookups

- **Cyclon - Successful lookups**
  - System size: 1,000, 1,100, 2,000
  - Successful lookups (%)
    - Desktop: green line
    - No desktop: red line

- **Kademlia - Successful lookups**
  - System size: 1,000, 1,100, 2,000
  - Successful lookups (%)
    - Desktop: green line
    - No desktop: red line

- **NCyclon - Successful lookups**
  - System size: 1,000, 1,100, 2,000
  - Successful lookups (%)
    - Desktop: green line
    - No desktop: red line

- **Kelips - Successful lookups**
  - System size: 1,000, 1,100, 2,000
  - Successful lookups (%)
    - Desktop: green line
    - No desktop: red line
Influence of desktop nodes

**Metric:** average number of messages per lookup

- **Cyclon - Messages per lookup**
  - System size: 1,000, 1,100, 2,000
  - Messages/lookup: 0, 2,000, 4,000, 6,000, 8,000, 10,000, 12,000
  - Desktop vs. no desktop

- **Kademlia - Messages per lookup**
  - System size: 1,000, 1,100, 2,000
  - Messages/lookup: 0, 2,000, 4,000, 6,000, 8,000
  - Desktop vs. no desktop

- **NCyclon - Messages per lookup**
  - System size: 1,000, 1,100, 2,000
  - Messages/lookup: 0, 1,000, 2,000, 3,000
  - Desktop vs. no desktop

- **Kelips - Messages per lookup**
  - System size: 1,000, 1,100, 2,000
  - Messages/lookup: 2.8, 3.1, 3.4
  - Desktop vs. no desktop
Influence of desktop nodes

**Metric**: number of messages needed for system management.
Algorithm memory footprint

Data structures size in memory (estimated)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory footprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclon</td>
<td>12.6 KBytes</td>
</tr>
<tr>
<td>Kademlia</td>
<td>102.5 KBytes</td>
</tr>
<tr>
<td>NCyclon</td>
<td>304.8 KBytes</td>
</tr>
<tr>
<td>Kelips</td>
<td>39.1 KBytes</td>
</tr>
</tbody>
</table>

Cyclon - Memory footprint
- Local view: 0.93%
- Handled lookups queue: 99.07%

Kademlia - Memory footprint
- Ongoing searches: 52.44%
- Buckets: 47.56%

NCyclon - Memory footprint
- Local view: 0.04%
- Lookup table: 99.96%

Kelips - Memory footprint
- Local view: 9.37%
- Contacts: 4.82%
- File tuples: 38.97%
- Gossiping stream: 46.84%
Protocol bandwidth usage

Bandwidth usage (estimated) on Scenario 1 with 100,000 profiles.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cyclon</strong></td>
<td>9,846 Kbit/sec</td>
</tr>
<tr>
<td><strong>Kademlia</strong></td>
<td>0,236 Kbit/sec</td>
</tr>
<tr>
<td><strong>NCyclon</strong></td>
<td>1,463 Kbit/sec</td>
</tr>
<tr>
<td><strong>Kelips</strong></td>
<td>36,825 Kbit/sec</td>
</tr>
</tbody>
</table>

**Bandwidth Distribution**

- **Cyclon** - Bandwidth distribution
  - 95,05% LOOKUP
  - 4,95% SHUFFLE

- **Kademlia** - Bandwidth distribution
  - 67,68% HEARTBEAT
  - 11,10% STORE
  - 11,10% JOIN_REQUEST
  - 11,10% JOIN_RESPONSE

- **NCyclon** - Bandwidth distribution
  - 82,77% ADVERTISE
  - 17,17% LOOKUP
  - 0,05% SHUFFLE

- **Kelips** - Bandwidth distribution
  - 99,94% HEARTBEAT
  - 0,06% STORE
  - 0,00% JOIN_REQUEST
  - 0,00% JOIN_RESPONSE
  - 0,00% LOOKUP