The GNUnet DHT

Christian Grothoff
christian@grothoff.org
http://grothoff.org/christian/

“I trust no one, not even myself.” –Joseph Stalin
Agenda

• A Quick Introduction to Bloom Filters
• The $R^5N$ Routing Algorithm
• Performance Analysis for $R^5N$
• Content validation
• The DHT API
• The BLOCK API
Bloom Filters

- Probabilistic data structure to answer the question “is element $X$ in set $S$” with “no” or “maybe”

- If an element is not in the set, the probability is high that the answer is “no”

- Uses a bit-array where $k$ bits based on $H(X)$ are set to 1 for each element $X \in S$. 
Review: Kademlia
Kademlia and Restricted Routes

Connections
Route path
Connection does not exist
The $R^5N$ Routing Algorithm

- Designed to work well in restricted route networks (many nearest peers) and reduce the impact of malicious peers.
- Requires recursive routing; less control for initiator, better performance; stateful return routing.
- Kademlia style routing table — so-called “$k$-buckets” storing $k$ peers; such that the $i^{th}$ $k$-bucket stores peers with $XOR$ distance between $[2^i, 2^{i+1})$.
The $R^5N$ Routing Algorithm

- Random and Kademlia style routing phases
  ⇒ combines path diversity with efficient routing
  - Random phase: “start” Kademlia routing from random location.
  - Kademlia phase: efficiently find nearest peers.

- Requests have desired replication level $r$; the number of nearest peers a request should reach.

- Achieved by probabilistic path branching, at each hop a request may be forwarded to one or more peers.
The $R^5N$ Routing Algorithm
The \( R^5 N \) Routing Algorithm

**PUT Request**

\[
\text{if nearest}(r) \text{ then} \\
\quad \text{store}_\text{data}(r) \\
\text{else} \\
\quad \text{for } i = 0 \rightarrow \text{num}_\text{forwards}(r) \text{ do} \\
\quad \quad p = \text{get}_\text{forward}_\text{peer}(r) \\
\quad \quad \text{forward}_\text{request}(r, p) \\
\quad \text{end for} \\
\text{end if}
\]

**GET Request**

\[
\text{if } \text{NULL} \neq (d = \text{find}_\text{data}(r)) \text{ then} \\
\quad \text{route}_\text{result}(r, d) \\
\text{end if} \\
\text{for } i = 0 \rightarrow \text{num}_\text{forwards}(r) \text{ do} \\
\quad p = \text{get}_\text{forward}_\text{peer}(r) \\
\quad \text{store}_\text{route}(p, r) \\
\quad \text{forward}_\text{request}(r, p) \\
\text{end for}
\]
Routes with Loops

- $R^5 N$ cannot loop forever due to the hop counter
- Looping is still inefficient

$\Rightarrow R^5 N$ uses a Bloom filter to avoid loops
Performance Analysis for $R^5N$

- Randomized routing takes $c$ steps, $c \sim \log n$
- Kademlia-style routing takes $O(\log n)$ steps

$\Rightarrow$ Finding a nearest peer is $O(\log n)$
Performance Analysis for $R^5N$

- There are $\frac{|N|^2}{|E|} \in O(|N|)$ nearest peers

- For a 50% success rate for a single GET, we need $O(\sqrt{|N|})$ replicas

- Then repeat GET $O(\sqrt{|N|})$ times for “high” success rate

⇒ Total routing cost is $O(\sqrt{n \log n})$
## Absolute Performance

<table>
<thead>
<tr>
<th>Size of network</th>
<th>Average hops per PUT</th>
<th>Average hops per GET</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R-Kademlia</td>
<td>$R^5 N$</td>
</tr>
<tr>
<td>100</td>
<td>2.70 ± 0.06</td>
<td>3.96 ± 0.06</td>
</tr>
<tr>
<td>250</td>
<td>3.06 ± 0.10</td>
<td>4.26 ± 0.10</td>
</tr>
<tr>
<td>500</td>
<td>3.08 ± 0.46</td>
<td>4.38 ± 0.45</td>
</tr>
<tr>
<td>750</td>
<td>3.19 ± 0.74</td>
<td>4.37 ± 0.83</td>
</tr>
<tr>
<td>1000</td>
<td>3.63 ± 0.07</td>
<td>4.47 ± 0.93</td>
</tr>
</tbody>
</table>
The DHT API

- GNUNET_DHT_connect, GNUNET_DHT_disconnect
- GNUNET_DHT_put, GNUNET_DHT_put_cancel
- GNUNET_DHT_get_start, GNUNET_DHT_get_stop
- GNUNET_DHT_monitor_start, GNUNET_DHT_monitor_stop
Special GET Options

GET requests can be given the following optional options:

- Bloom Filter: filter known results (duplicates)
- Bloom Filter Mutator: change hash function of Bloom Filter
- eXtended Query: additional query information beyond the hash
Options for GET and PUT

- GNUET_DHT_RO_DEMULTIPLEX_EVERYWHERE
- GNUET_DHT_RO_RECORD_ROUTE
- Replication level
- Expiration time (provided to PUT, returned by GET)
- Block type ⇒ for content validation
Monitoring

DHT monitoring is useful for...

- Testing / debugging
- Performance analysis
- Application development!
The BLOCK API

• Block type determines responsible Block plugin

• Configuration option `[block] PLUGINS` specifies supported plugins

• Implement a new plugin based on the `gnunet_block_plugin.h` header

• “fs” for file-sharing, “dht” for DHT internals, “test” for no verification (any data can match any key)
The BLOCK Plugin API

Each plugin must provide two functions:

- **GNUNET_BLOCK_EvaluationFunction**: does the given block satisfy the requirements of the given query? Possible answers include: Yes, and other replies can exist; yes, and this is the only answer; no, duplicate reply; no, invalid reply

- **GNUNET_BLOCK_GetKeyFunction**: given a block, what key should it be stored under? Possible answers are: A key; bad block; not supported
Experimental Results: Replication

The graph shows the total number of replicas in the network across different rounds of PUTs for R5N Random, R-Kademlia Random, R5N Same, and R-Kademlia Same.
Experimental Results: Sybils

The figure shows a graph with the x-axis representing the number of malicious peers in the network and the y-axis representing the GET success percentage. The graph compares different rounds and algorithms:

- R5N Round 10
- R5N Round 5
- R5N Round 1
- R-Kademlia All Rounds

The graph illustrates how the success percentage decreases as the number of malicious peers increases.
DHTs and your Projects

- Distributed Search:
  - Distributing sites to spider?
  - Distributing search queries & replies?

- Jugendschutz:
  - Finding rankings?
  - Finding culturally close users?

- Twitter Censorship:
  - Finding peers monitoring similar feeds?
  - Finding missing tweets?
Copyright

Copyright (C) 2011, 2012 Christian Grothoff

Verbatim copying and distribution of this entire article is permitted in any medium, provided this notice is preserved.